1883

ARTIFICIAL INTELLIGENCE PROJECTS FOR THE COMMODORE 64™

TIMOTHY J. O'MALLEY

VISIT…

LANZAROTE
Caliente.COM

ARTIFICIAL INTELLIGENCE PROJECTS FOR THE COMMODORE 64™

ARTIFICIAL INTELLIGENCE PROJECTS FOR THE COMMODORE 64™

TIMOTHY J. O'MALLEY

TAB

TAB BOOKS Inc.

Blue Ridge Summit, PA 17214

FIRST EDITION

SECOND PRINTING

Printed in the United States of America

Library of Congress Cataloging in Publication Data

O'Malley, Timothy J.
Artificial intelligence projects for the
Commodore 64.

Includes index.
1. Artificial intelligence—Data processing.
2. Commodore 64 (Computer)—Programming. I. Title.
Q336.043 1985 001.53'5 84-26749
ISBN 0-8306-0883-4
ISBN 0-8306-1883-X (pbk.)

Contents

Programs in the Book

Introduction

This book is more than just a collection of artificial intelligence programs and their descriptions. It is a source book of ideas concerning how to solve problems. It is a guide to how humans solve problems and how the Commodore 64 might be programmed to do so. It is written so that you might become a better programmer and might explore different ways of solving problems. I have done my best to design programs that will display important aspects of artificial intelligence on the Commodore 64.

Think about artificial intelligence for a moment. Let's not fool anyone; artificial intelligence is a set of ways to make the computer make better use of the information that it contains. All of the rules of computer science apply to the field of artificial intelligence. Don't expect that you can somehow transform your Commodore 64 into an electronic personality. In this book I will not attempt to answer the question of whether or not computers can be programmed to think. I sometimes ask people if people can think. Human intelligence may also be an illusion.

In this book you will look at ways that the Commodore 64 can be used to make deductions and solve problems. In some programs, the computer "learns" strategy from a game-playing opponent, you. It then uses that stored strategy to make a move when it encounters the same situation. Is this learning? It is learning in an artificial sense and may not utilize the same mechanics as human learning. Other programs change themselves or the data that they contain. These program might increase in complexity. In any artificial intelligence program, the acid test is whether or not the computer will respond better with time. Much of artificial intelligence simply involves getting the data organized so that the computer can access it in a coordinated manner.

This book is written for both the beginner and the seasoned programmer. The beginning programmer can run the programs even though he or she may not understand all of the operations and techniques involved. The experienced programmer will find some surprises, mostly in the ways of approaching problems and in the ways BASIC statements can be arranged. This book then is really appropriate for a wide range of programmers, although some fundamental understanding of computing is essential. Home computer owners who are interested in artificial intelligence will find that this book satisfies their curiosity. This book would also be good as outside reading or as an added text for a math or computer science course. Those in psychology courses in which computers are used will find the book of interest.

I hope that you will use this book in a constructive manner. I hope that the programs in it will help you to look at problems and thinking in a fashion that you hadn't considered previously. At the present stage of computer technology, the lagging area is software development. This book will help you develop a proper perspective of artificial intelligence and computer software generally.

Chapter 1

An Introduction to Artificial Intelligence

In this first chapter I introduce some definitions and techniques associated with artificial intelligence. You will see some of the kinds of things that can be done on the Commodore 64 microcomputer. Two introductory programs that exhibit artificial intelligence to some degree are presented.

WHAT IS ARTIFICIAL INTELLIGENCE?

At the risk of being in trouble before I start, let me try to define a working definition of artificial intelligence. This definition may not be inclusive but it is a start.

DEFINITIONS AND SCOPE

Artificial intelligence is the attempt to set machines to perform processing abilities normally associated only with cognitive human thought. It is artificial in the sense that it is a manufactured technique and may not have the same basis as human intelligence. What is important is that the results of human and artificial intelligence are highly similar. Artificial Intelligence is really more than that, though. It is the ability of the computer to better utilize the information that it contains. For example, if you somehow store the fact that Socrates is a man in the computer and then store the fact that man is mortal, you might be able to devise an algorithm such that the computer can deduce that Socrates is mortal. Thus the computer might be able to solve the hypothetical syllogism.

Much of how the computer can solve problems lies in how the data is organized in memory. I will use set theory in some of my programs. If I say that Socrates is a proper subset of man and if I say that man is a proper subset of things mortal, it follows logically that Socrates is mortal. If set A is contained in set B, and set B is contained in set C, then set A is contained in set C. I will design a truth table for solving problems. Then the computer can use this truth table to solve all problems that relate to set theory.

Artificial intelligence, in my opinion, is really an illusion. In fact, I think that thinking may be an illusion. If these words that you are reading don't seem to contradict the facts that you know already, you might say that an intelligence had written them. We might contrive a survey such that people would be polled to see if certain computer abilities were indistinguishable from human abilities. If they could not tell the difference, we might say that the computer exhibited artificial intelligence. Such a test is called the *Turing* test, after Alan Mathison Turing.

Turing, more than anyone else, laid the mathematical framework for digital computers. Turing machines are the logical basis of every digital electronic computer in existence. Turing machines are conceptual devices that allow for the binary solution of problems in electronic computers. Turing's work has led me to believe that any problem that can be defined as a set of binary components is ultimately solvable. This makes me think that binary arithmetic is fundamental to the problem-solving process.

The goal of artificial intelligence is to be able to program a computer to solve any given set of problems in a reasonable period of time. Although some problem might be theoretically solvable, the time involved might be astronomical. Therefore we have to look at other ways of solving problems. I hope that this book will challenge us to explore other ways of solving problems.

There are a number of areas that artificial intelligence encompasses. One is *natural language processing*. In this area the computer responds in English or another language and "understands" commands and statements in that language.

Another area is *expert systems*. In expert systems the computer is able to respond to many different commands or questions about a limited discipline. It is an expert.

Another area is problem-solving, in which the computer attempts to find patterns or rules in order to solve problems. Maybe it will store data from prior problems and then see if there is a solution based on past information. We might even supply the rule so that the computer can solve the problem.

Another area is behavior where we can "train" the computer to perform a certain "learned" sequence of operations. If a step in that sequence produces a negative effect, we might have the computer try another possible operation. Eventually we can have the computer "trained" to do the desired sequence of events.

In *heuristics* the computer may look for the best operation at a given point in time. If that operation turns out to be wrong, it will "remember" not to do that again. Many of these different areas overlap and a program may not lie strictly in one area.

Microcomputer Applications

Let's be honest. When we think of artificial intelligence, we generally think of cryogenic supercomputers operating at the limits to technology, or at least of a large mainframe using a high-level language like LISP. Is it possible to run artificially intelligent programs on a microcomputer programmed in BASIC or machine language? I think so. At least some problems can be run on a micro if those problems do not use an excessive amount of memory or time.

The Commodore 64 is a good choice for some artificial intelligence programs. It has 64 K RAM and can store data on disk or tape. A disk drive will allow fast access to files of information. These files can be updated as a program is run. Thus the storage capacity is large enough to do some modest problem-solving. The cost is low enough to enable many people interested in artificial intelligence to conduct their own experiments. The speed of the Commodore is reasonably fast. For those problems requiring more speed, the programmer might resort to machine language instead of interpreted BASIC. Machine language is easily accessed on the Commodore 64. The programmer might also use a BASIC compiler to translate his program into faster code.

ARTIFICIAL INTELLIGENCE TECHNIQUES

We now look at four techniques used in artificial intelligence. They are tree searches, the algorithmic (rule) method, the heuristic method, and pattern searching. We realize other methods may exist, but these are some of the techniques best suited to be explored using the Commodore 64.

Tree Searches

A *tree* is a graph made of points, called *nodes,* and connecting lines. They are designed so that the number of nodes increases as you go from top to bottom. Actually the tree resembles an inverted tree structure with many branches. If you visualize your family tree, you might see a few ancestors at the top, and many levels of descendants as you go toward the bottom. Genealogical diagrams are a form of tree.

Another type of tree reflects the number of possible moves in a game. As the move number increases, the possible outcomes of the game increases. For example in the game of chess, you start with 20 possible moves for move number one by WHITE. Any of the eight pawns can move one or two squares forward, or either of the two knights can jump to one of two possible squares. BLACK has the same number of possible moves on its first move. After that the possibilities start to multiply rapidly.

A tree search would go down all the levels (in this case move numbers) to find all the possibilities. The computer would check out the possible outcomes of the moves with a tree search.

A complete search. In a complete search, the computer would explore the outcomes of every possible move in a game. It would then take those moves that would assure a victory. In the game of chess, a complete search would be impossible because the number of possible games are virtually infinite. A complete tree search would require an infinite amount of time.

In some simple games, like tic tac toe, a complete search would make sense because the possible games are a small finite number, nine factorial, or 362,880. Actually it is smaller than that if we disregard move numbers and only consider positions as combinations of moves. Games are probably the most obvious examples of where tree searches are used.

A pruned search. Because a complete search is impossible for some situations, such as the game of chess, we can employ a modified tree search called a pruned search. In a pruned search, we eliminate those moves that are clearly nonsense moves, moves that lead to immediate defeat. We save computer time by searching only those moves that seem to have a good possibility of leading to a victorious outcome. In the minimum-maximum prune search, we take those moves that give us the minimum amount of danger of losing and that our opponent could use to gain the maximum advantage. Thus we try to find our best move and try to predict how our opponent will move. We probably set some time or level limit for this pruned search and then choose the move that gives the best position at the lowest level. Many typical microcomputer programs for chess or checkers only go down a few levels in a tree search. Tree searches, then, can be useful for finding best moves in some games.

The Algorithmic Method

In the algorithmic method, we use a rule or algorithm to solve a problem. For example if you are blind and lost in a maze, you might find your way out of the maze by running your left or right hand along the wall. Eventually you will trace your way through the maze to find the exit. This may not be the fastest method, but it works. Another method for finding your way out of a maze is to somehow mark all dead ends. As you successively mark dead ends, you effectively debranch the maze. The resulting trunk is the solution and leads to the exit. Obviously you may well find your way out of the maze before you have debranched all of the dead ends. So sometimes a simple rule can lead to the solution, no matter what the problem.

Often the solution is a simple binary solution. In the first two programs in this chapter, you will see how a simple binary rule can solve the Towers of Hanoi puzzle. In that puzzle you move disks of varying diameters on three different posts. The rules of the puzzle states that you can move only one disk at a time and can not place a larger disk on top of a smaller disk. When we discuss that program we will see how a simple binary rule can tell us where to place which disk, regardless of the number of disks. The game of Nim may have a binary solution as well.

The Heuristic Method

Heuristics can include such things as the pruned search, where the computer ignores exploring dead end possibilities. Let's expand the definition of heuristics to include any process whereby the computer will debranch a tree or maze, or will reduce the probability of using certain routes. In a game we can set up a probability array corresponding to certain moves. If the computer makes certain moves that leads to a defeat, we might reduce the numbers in the probability array. The computer would then choose from those move numbers that have a higher probability of success. In this way the computer might successively approximate the correct sequence of moves leading to a win. We might say that the computer has learned strategy from past games. Likewise the probability of choosing those numbers that lead to victory might be increased so that the computer would choose them more often.

Pattern Searching

In pattern searching the computer looks for a sequence of numbers or a complicated rule to solve the problem. For example if we stored the notes from many songs in memory and then asked the computer to name the tune after just a few notes, the computer would search through all the note sequences until it found a match. Then it might give the title of that song. Or we might have games stored on file and ask the computer when a combination of moves were made that lead to a victory. We would have the computer look for patterns.

Binary patterns. We said earlier that we suspected that binary arithmetic may be fundamental to the problem-solving process. If a binary pattern could be found, we might have understood something about the nature of the problem. Binary patterns are the simplest form of language, although it might not seem so to us. (Consider the firefly: its on-off flashes represent a binary form of communication. Other fireflies have no problem in deciphering the message.) This may become more obvious later.

Other patterns. Many times there is no clear-cut rule, only the sequence itself. Consider the number of days in a month. We would either have to invent a rule to remember or look at a calendar. There really is no simple mathematical rule to relate days of the month to the name of the month or vice versa. As we said, sometimes we make up a rule so that we can remember. You might think about how a computer could devise a sequence rule to solve problems after it has found the pattern initially.

Other times the pattern can best be expressed in a non-binary, mathematical way. Perhaps the relationship between two given variables is a simple polynomial expression. If we use statistics, we might find a polynomial regression line that best relates the two variables in question. Or let's say we want the computer to distinguish the difference between men and women based on height and weight. For the sake of argument, let's say that men tend to be taller and heavier than women. We would have the computer plot a height-weight graph. We would then tell the computer which points were male and which were female. The male points would probably cluster together and the female coordinates would cluster together. If we entered a height and weight and asked the computer whether it was male or female, it would choose whichever cluster of points were nearest in making its determination. The program's data would be constantly updated. Statistics would shed some light on the mathematics of the problem. We might also want to include another factor, such as age, and then plot in three dimensions to predict the sex.

In this book I will describe all of these ways of solving problems. More than a dozen programs will be examined in depth and every line will be explained in detail. I will suggest ways to alter the programs. This book assumes that you own or have access to a Commodore 64 computer with a C2N or equivalent tape recorder/player. The programs can also be used with disk drives. We will look at the different kinds of programs that can be used in artificial intelligence experiments. This book will serve as a starting point for those interested in artificial intelligence. It doesn't presume to cover every area of artificial intelligence. Discoveries of new ways of using computers are being made each day.

THREE INTRODUCTORY PROGRAMS IN BASIC

The three programs that follow display important aspects of artificial intelligence. The first two programs are similar. They solve the Towers of Hanoi puzzle. The first produces a list of the moves that must be made to solve the puzzle, the second program shows the solution graphically. The third program shows the solution to the knight's tour problem. In the knight's tour, the object is to have a knight in the game of chess land on all 64 squares of a chessboard without landing on a square more than once.

The Towers of Hanoi: Version One

The challenge in this puzzle is to move all the disks from one of three posts to another of the posts, without placing a larger disk on top of a smaller disk. Only one disk can be moved at a time. Listing 1-1 shows a unique binary solution to the puzzle.

Listing 1-1 The Towers of Hanoi: Version 1

```
5 :::::::::::::::::::::::::::::::::::::
10 REM   TOWERS OF HANOI - VERSION 1
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
45 :::::::::::::::::::::::::::::::::::::
50 PRINT CHR$(147);
60 INPUT "WHAT NUMBER OF DISKS";N:M=(1=(1 AND N))*2+1
70 DIM K(N):FOR I=1 TO N:K(I)=1:NEXT
80 A$="ABC":FOR L=1 TO 2↑N-1:I=0
90 D=-((L AND 2↑I)=2↑I)*(I+1):I=I+1:IF D=0 THEN 90
100 T=M*(((1 AND D)=1)*2+1)
110 F=K(D)-T:J=F-T*3*(F<1 OR F>3)
120 PRINT "MOVE"L":"TAB(15)"FROM "MID$(A$,K(D),1)" TO ";
130 PRINT MID$(A$,J,1):K(D)=J:NEXT
135 :::::::::::::::::::::::::::::::::::::
```

Here's how the solution works. First, let's number the disks: the smallest disk on top will be assigned the number 1; the disk beneath that will be called 2, and so forth down to N number of disks. Second let's number the posts (or towers): the A post, which is the post holding all the disks initially, is 1; the B post is 2, and the C post is 3. Third, the total number of moves is 2 to the power of N minus 1. L is defined as the move number variable.

The variable M is set to 1 when the number of disks is even and to −1 when the number of disks is odd. This is all the information that is needed to solve the problem.

When we represent the move number as a binary number, it becomes evident what disk to move; the disk number is a binary function of the move number! The disk to move reflects the position of the rightmost unity (1) bit of the move number. If L is 0001, we move disk 1; if L is 0010, we move disk 2; if L is 0100, we move disk 3; if L is 1000, we move disk 4; if L is 1100, we move disk 3.

We can determine what post to move that disk to as a function of the variable M. If M is −1, the odd-numbered disks go to posts in the sequence 3 to 2 to 1 to 3 to 2 to 1 and so on; the even-numbered disks go to posts in the sequence 1 to 2 to 3 to 1 to 2 to 3 and so on. If the variable M is 1, the even-numbered disks go according to the first sequence and the odd numbered disks go according to the second sequence. The problem is solved when L has a final value of 2 to the power of N minus 1.

Here is a detailed explanation of each program line.

LINE	**EXPLANATION**
5	The colon is a nonexecutable character in BASIC. It is used here to outline the edge of the program listing.
10	This REM (for REMark) is not executed in BASIC. It simply acts as a comment statement in BASIC. In this case it tells the name of the program. It's usually a good idea to use REM statements to identify and explain parts of your program.
20	This REM statement identifies the programmer/author.
30	This statement indicates the copyright ownership of the program. TAB BOOKS, Inc.
40	This statement says that the program is written for the Commodore 64.
45	This set of colons separates the header information from the main body of the program.
50	This line clears the screen. CHR$(147) is the command code for the clear screen command. The semicolon (;) keeps print position from jumping to the next line after the PRINT statement is executed. Thus the next printing will start at left side of the top line.
60	This line is made of two statements separated by a colon. The first statement requests the value of N after it prints the question, "WHAT NUMBER OF DISKS?" INPUT statements sometimes contain prompting information when they ask for values. The second statement on this line defines the value of M. 1 AND N will give a value of 1 if N is odd, and a

value of 0 if N is even. 1=1 will give a value of −1, meaning true. 1=0 will give a value of 0, meaning false. These are Boolean algebra logic statements. We multiply the earlier result by 2 and add 1 to get the value of M. Note the double use of the equal sign (=) in this statement: the equal sign is used both to perform a logical test and to assign the value of the expression to the variable M.

70 DIM K(N) assigns space in memory for a list of numbers that we will call K. K is an array with N number of elements. The next three statements in this line form a loop that sets the value of each of these N elements to 1. Loops are common in virtually all programming languages. Here we make up a variable called I that will have values that will range from 1 to N, whatever N is. The NEXT statement says to continue this loop until I exceeds N. We could also have said NEXT I instead of simply NEXT, but NEXT by itself is somewhat faster.

80 This line assigns the string variable, A$, to ABC. A$ is like the K array, only it contains individual characters instead of numerical values. We then start an L loop, with the values of L ranging from 1 to 2 to the power of N minus 1. L is the number of moves that are required to solve the puzzle. The variable I is set to zero.

90 This line determines what disk to move as a function of the move number, L. The value of I is increased by 1, and if D equals 0, the line is rerun. Eventually D will reach a nonzero value.

100 The value of T is assigned as a function of M and D.

110 F is assigned to K at D minus T. J is set as a result of the algebraic and logical expression (F<1 OR F>3), which is evaluated as either 0 or −1.

120 This line prints the move number, spaces over to the fifteenth print position, and then prints the post that the disk is moved from, A, B, or C. These letters are printed using the MID$ function.

130 This line prints the letter of the post that the disks are moved to. Then the array K at the element D is set to the value of J. The L loop is terminated at this line.

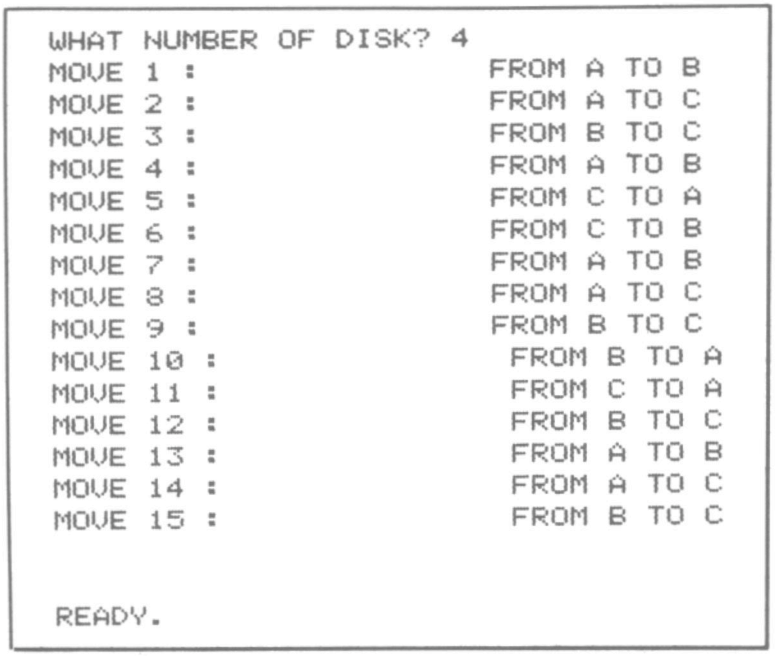
135 This line of colons gives a boundary to mark the end of the program listing.

Program Operation. Figure 1-1 shows the result of moving four disks. Figure 1-2 shows the result of moving five disks. Notice that the number of steps doubles with each additional disk. The statements in this program have been written for a minimum amount of computation, and the program starts to print immediately, regardless of the number of disks involved. It is interesting to note that if you know the disk numbers and the move number, you can immediately deduce the proper disk to move and the place to move it, regardless of the number of disks used in the problem.

Towers of Hanoi: Version Two

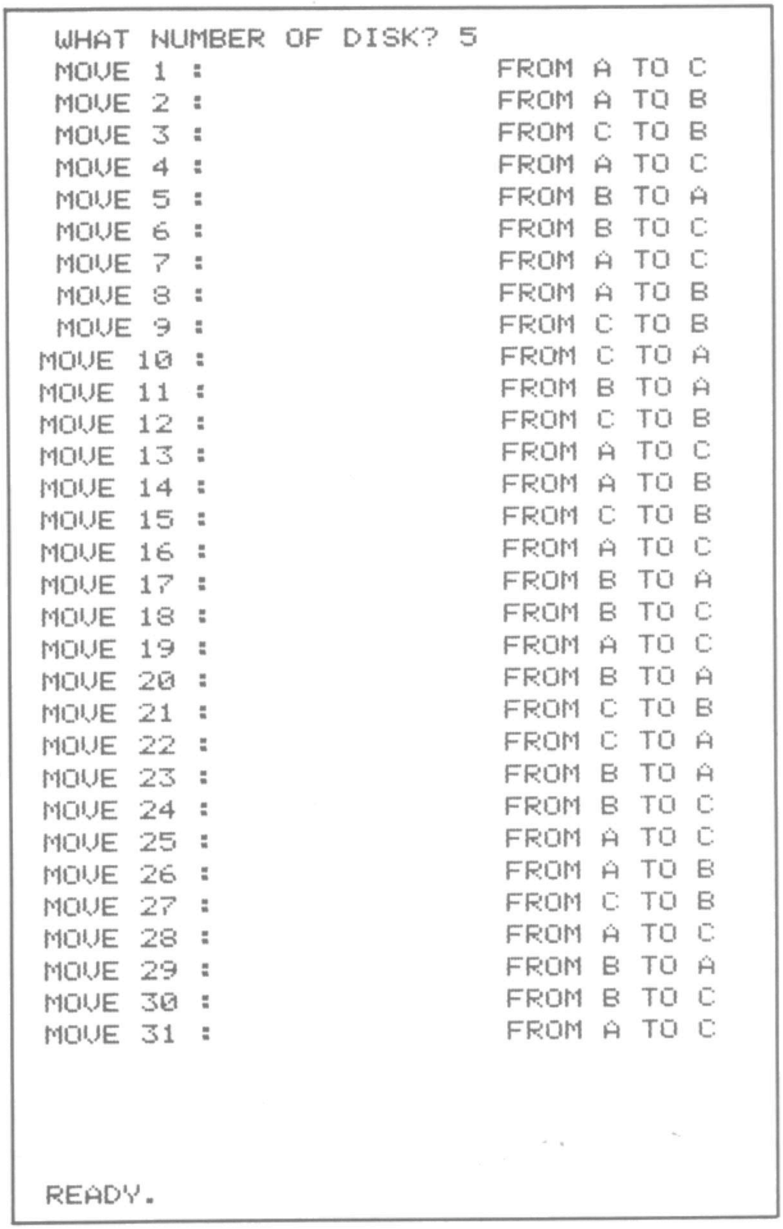
In the version of the Towers of Hanoi puzzle, shown in Listing 1-2, we display a graphic solution, using sprites as disks. These sprites are like user-defined graphic characters and can be of several colors. We use a different color for each sprite to help distinguish them from each other. Each sprite is of a different length, showing the different sizes of the disks.

This program lets you have the computer use as many as eight disks in the puzzle. There is a limit of eight disks because there are only eight sprites, numbered from 0 to 7. That means that the computer would make up to 255 moves to solve a puzzle involving all of the sprites.



```
WHAT NUMBER OF DISK? 4
MOVE 1 :               FROM A TO B
MOVE 2 :               FROM A TO C
MOVE 3 :               FROM B TO C
MOVE 4 :               FROM A TO B
MOVE 5 :               FROM C TO A
MOVE 6 :               FROM C TO B
MOVE 7 :               FROM A TO B
MOVE 8 :               FROM A TO C
MOVE 9 :               FROM B TO C
MOVE 10 :               FROM B TO A
MOVE 11 :               FROM C TO A
MOVE 12 :               FROM B TO C
MOVE 13 :               FROM A TO B
MOVE 14 :               FROM A TO C
MOVE 15 :               FROM B TO C


READY.
```

Fig. 1-1. The solution to the Towers of Hanoi puzzle with four disks.



```
 WHAT NUMBER OF DISK? 5
 MOVE 1 :               FROM A TO C
 MOVE 2 :               FROM A TO B
 MOVE 3 :               FROM C TO B
 MOVE 4 :               FROM A TO C
 MOVE 5 :               FROM B TO A
 MOVE 6 :               FROM B TO C
 MOVE 7 :               FROM A TO C
 MOVE 8 :               FROM A TO B
 MOVE 9 :               FROM C TO B
MOVE 10 :               FROM C TO A
MOVE 11 :               FROM B TO A
MOVE 12 :               FROM C TO B
MOVE 13 :               FROM A TO C
MOVE 14 :               FROM A TO B
MOVE 15 :               FROM C TO B
MOVE 16 :               FROM A TO C
MOVE 17 :               FROM B TO A
MOVE 18 :               FROM B TO C
MOVE 19 :               FROM A TO C
MOVE 20 :               FROM B TO A
MOVE 21 :               FROM C TO B
MOVE 22 :               FROM C TO A
MOVE 23 :               FROM B TO A
MOVE 24 :               FROM B TO C
MOVE 25 :               FROM A TO C
MOVE 26 :               FROM A TO B
MOVE 27 :               FROM C TO B
MOVE 28 :               FROM A TO C
MOVE 29 :               FROM B TO A
MOVE 30 :               FROM B TO C
MOVE 31 :               FROM A TO C



READY.
```

Fig. 1-2. The solution to the Towers of Hanoi puzzle with five disks.

This program uses the same binary solution to solve the Towers of Hanoi puzzle as the first program did. It starts to operate by moving the disks almost immediately. Other computer programs solve this problem by using many recursive techniques, but none is quite so simple as the one employed in this program.

Listing 1-2 The Towers of Hanoi with Graphics

```
10 ::::::::::::::::::::::::::::::::::::::
15 REM          TOWERS OF HANOI
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
25 REM COPYRIGHT 1983, TAB BOOKS INC.
30 REM (WRITTEN FOR THE COMMODORE 64)
35 ::::::::::::::::::::::::::::::::::::::
40 REM       *** MAIN PROGRAM ***
50 GOSUB 100: REM PRINT INSTRUCTIONS
60 GOSUB 300:REM DEFINES SPRITES
70 GOSUB 500:REM DETERMINE SOLUTION
80 END
90 ::::::::::::::::::::::::::::::::::::::
100 REM   *** PRINT INSTRUCTIONS ***
105 NM=0
110 PRINTCHR$(147);:REM CLEAR SCREEN
120 PRINT"   ARTIFICIAL INTELLIGENCE - PROGRAM 1"
130 PRINT:PRINT"           THE TOWERS OF HANOI":PRINT:PRINT
140 PRINT"      THE PROBLEM IS TO MOVE ALL OF THE"
150 PRINT"DISKS FROM THE FIRST POST TO THE THIRD"
160 PRINT"POST, ONE DISK AT A TIME, WITHOUT "
170 PRINT"PLACING A LARGER DISK ON TOP OF A "
180 PRINT"SMALLER DISK.":PRINT
190 PRINT"      YOU MAY SPECIFY 1-8 DISKS.":PRINT
200 OPEN1,0
210 PRINT"ENTER NUMBER OF DISKS: ";:INPUT#1,N$
220 PRINT:CLOSE1
230 IF LEN(N$)>1THENPRINT:GOTO190
240 N=VAL(N$):IFN>8ORN<1THENPRINT:GOTO190
250 PRINT:PRINT"VERY WELL."
260 RETURN
270 ::::::::::::::::::::::::::::::::::::::
300 REM    *** DEFINE SPRITES ***
305 POKE53281,15:REM BORDER GRAY3
310 V=53248:POKEV+32,15:IF N=8 THEN 320
315 FORS=NTO8:POKEV-2+2*S,0:POKEV-1+2*S,0:NEXTS
320 FORS=1TON:POKEV+21,PEEK(V+21)OR(2↑(S-1)):REM ENABLE SPRITES
330 POKEV+23,PEEK(V+23)OR(2↑(S-1)):REM EXPAND IN Y DIRECTION
340 POKEV+29,PEEK(V+29)OR(2↑(S-1)):REM EXPAND IN X DIRECTION
350 POKEV+37+2*S,S+5:REM SET SPRITES' COLORS
360 POKEV-2+2*S,50:REM SET EACH SPRITE TO X COORDINATE AT POST 1
370 POKEV-1+2*S,200:REM SET EACH SPRITE TO Y COORDINATE AT POST 1
375 VL=191+S
380 POKE2039+S,VL:REM SET SPRITE'S POINTERS
385 FORK=64*VLTO64*VL+63:POKEK,0:NEXTK
390 FORK=64*VL+6*(S-1)TO64*VL+6*(S-1)+3STEP3
400 POKEK,2↑S-1
410 POKEK+1,255
420 POKEK+2,255-(2↑(8-S)-1)
430 NEXTK,S
440 PRINTCHR$(147);:REM CLEAR SCREEN
```

```
450 FORK=1TO15:PRINTCHR$(17);:NEXTK
460 FORK=1TO8:PRINTTAB(6);CHR$(98);TAB(18);CHR$(98);
470 PRINTTAB(30);CHR$(98)
480 NEXTK:PRINT:PRINTTAB(6);"1";TAB(18);"2";TAB(30);"3"
485 PRINTCHR$(19):RETURN
490 ::::::::::::::::::::::::::::::::::::::
500 REM  *** DETERMINE SOLUTION ***
510 T(1)=2↑N-1
520 NM=NM+1:S=0
530 D=-((NMAND2↑S)=2↑S)*(S+1):S=S+1:IFD=0THEN530
540 TF=0
550 TF=TF+1:IF0=((2↑(D-1)ANDT(TF))=(2↑(D-1)))THEN550
560 DT=(((1ANDN)=1)*2+1)*(((1ANDD)=1)*2+1)
570 TT=TF-DT-DT*3*((TF-DT)>3)OR(TF-DT<1))
580 T(TF)=T(TF)-2↑(D-1)
590 T(TT)=T(TT)+2↑(D-1)
600 GOSUB700:REM DISPLAY DISK MOTION
610 IF NM<2↑N-1THEN520
620 PRINT"DONE.":RETURN
630 ::::::::::::::::::::::::::::::::::::::
700 REM  *** DISPLAY DISK MOTION ***
710 FORY=200TO100STEP-1:POKEV-1+2*S,Y:NEXTY
720 FORX=(TF-1)*96+52TO(TT-1)*96+52STEPSGN(TT-TF)
730 POKEV-2+2*S,X:NEXTX
740 FORY=100TO200:POKEV-1+2*S,Y:NEXTY
750 RETURN
```

Here is a detailed explanation of the listing.

LINE	**EXPLANATION**
10–35	This is the title and credit information for the program.
40–80	This is the main program. It consists of three subroutine calls and an END statement. By writing the program as a series of subroutines, you can keep it better organized and easier to understand. Line 40 is a REM statement identifying the section as the main program. Line 50 is a subroutine call that prints the instructions and asks how many disks. Line 60 defines the *sprites*. Sprites are graphic objects that you can define and can move around the screen quickly. We will use sprites to create the disks that the computer will move. Line 70 calls the sub-

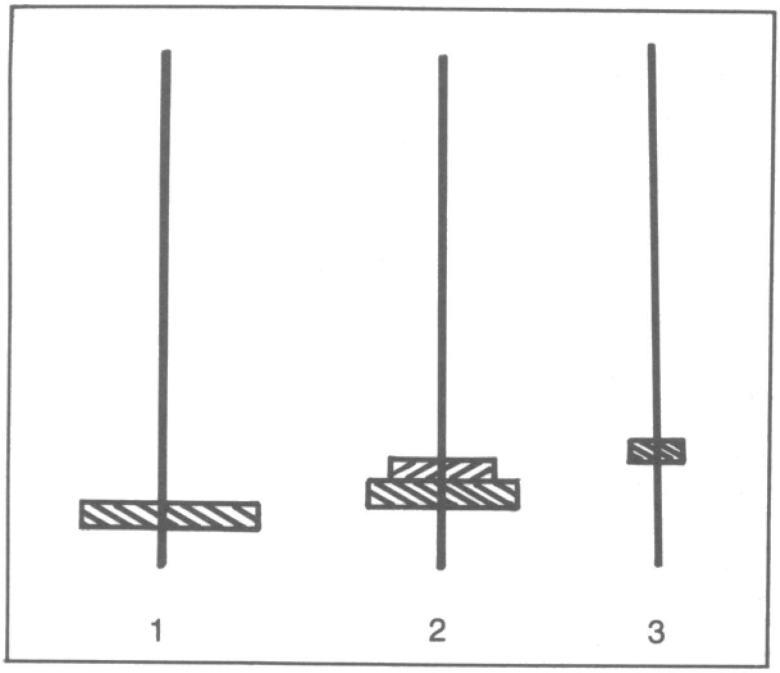


Fig. 1-3. The towers of Hanoi puzzle with four disks after move five.

	routine that determines the solution to the puzzle. Line 80 ends the program. Otherwise it would run into the subroutine that starts at line 100.
90	Colons are used to separate the main program from the first subroutine. They are nonexecutable.
100-260	These lines print the instructions and ask the user to input the number of disks. Line 200 is a command that allows the computer to read the keyboard, an alternative to using the INPUT statement. Line 210 prompts the user to enter the number of disks. The INPUT#1,N$ statement reads the keyboard and doesn't print a question mark on the screen. Line 220 ends the keyboard reading command. Actually it closes the keyboard "file." Line 230 checks for errors, as does line 240. Line 250 indicates that the number has been accepted. Line 260 returns the control back to line 60 of the main program.
270	The colons form a separation between the subroutines.
300-485	This subroutine defines the sprites.
300	This line identifies the subroutine.
305	This command changes the border color, the area around the edge of the screen, to color number 15,

	known as gray 3. POKE is a command used to change the values in individual memory locations.
310	V is a variable that is set to the value of the memory location at the beginning of the video chip register. The next POKE sets the rest of the screen to gray 3. If the number of disks (or sprites) is 8, control is sent to line 320.
315	This line is a loop that sets all the sprites at the initial position, off the screen. This is used to clear the screen.
320	The S loop enables the sprites in this line.
330	This line expands the sprites to double their original size in the vertical direction.
340	This line expands each sprite in the horizontal direction to twice its size.
350	This line uses the POKE command to set the color of each sprite.
360	This line moves each of the sprites used to its horizontal position on the first post.
370	This line positions each of the sprites to the proper vertical position.
375-430	These lines define the sprites' pointers and define the individual sprites. Each sprite is defined as a rectangle, which used to represent the disk. Parts of the sprites that are not defined are transparent to other sprites and whatever else that is on the screen.
440	This line clears the screen except for the sprites.
450	This line moves the cursor down 15 lines from the top.
460-485	These lines draw the vertical lines representing the posts and the letters A, B, and C. CHR$(19) is the code for the home character; it positions the cursor to the top left corner without clearing the screen.
490	This is a separator line.
500-620	This subroutine is essentially the same as was used in the first program. It determines the binary solution to the puzzle. Line 600 calls the subroutine starting at line 700, which shows the disks moving. Line 620 prints the word, "DONE" when the program is over.
630	This is a subroutine separator line.
700	This identifies the start of the disk (sprite) moving subroutine.
710	This line moves the sprite in question vertically from its resting place on the post. It is moved one pixel at a time, giving a smooth motion.

720-730	These lines move that sprite horizontally to the proper post, as determined by the subroutine starting in line 500.
740	This line drops the sprite down on the proper post.
750	This line returns control to line 610.

Program operation. After you have entered the program into RAM and typed RUN, the instructions will be displayed. The program will request the number of disks that you wish to have in a stack on the first post. You may specify an integer from 1 to 8. If you specify 1, the computer will solve the puzzle in 1 move. If you specify 8, the computer will solve the puzzle in 255 moves. As we said, the only real limit to the number of disks is the number of available sprites (8). If that weren't the limiting factor, then computer speed would be the limiting factor in solving the puzzle.

This program is rather interesting to watch because of the graphics. It looks like someone is moving the disks from one post to another. This program is visually more meaningful than the first program. Enjoy!

The Knight's Tour

The program in Listing 1-3 is a computer solution to the knight's tour puzzle, which requires that the knight land on each position on the chess board without landing on any position twice. Although there are many possible solutions for the knight's tour from any given starting position, this program displays one possible solution for each of the 64 possible starting positions. This program is something of an expert system because the computer is able to quickly solve the problem from any first position. A solution of the knight's tour is contained in DATA statements in lines 601-608. These data are rearranged to give other possible solutions. Thus the computer appears more "intelligent" than it is. Also the solution is displayed almost immediately.

This program, then, represents an unusual way of solving the knight's tour puzzle. Other methods have included the use of stacks to trace the knight's way through the chessboard. This program is particularly good if you don't care how the knight finds his way from a given starting position.

Here is an explanation of the program lines.

LINE	**EXPLANATION**
10-60	These lines identify the program.
70-110	This is the main program. Line 70 identifies the section. Line 80 is a subroutine call to line 200 to print the instructions. Line 90 is a subroutine call to

Listing 1-3 The Knight's Tour

```
10 :::::::::::::::::::::::::::::::::::::
20 REM          KNIGHT'S TOUR
30 REM WRITTEN BY TIMOTHY J. O'MALLEY
40 REM COPYRIGHT 1983, TAB BOOKS INC.
50 REM (WRITTEN FOR THE COMMODORE 64)
60 :::::::::::::::::::::::::::::::::::::
70 REM      *** MAIN PROGRAM ***
80 GOSUB 200:REM PRINT INSTRUCTIONS
90 GOSUB 400:REM DEFINE GRAPHICS
100 GOSUB 600:REM DISPLAY SOLUTION
110 OPEN 1,0:INPUT#1,N$:CLOSE 1:PRINTCHR$(147):POKE53281,6:POKES,0:
    POKES+1,0:END
120 :::::::::::::::::::::::::::::::::::::
200 REM     *** INSTRUCTIONS ***
210 PRINTCHR$(147);:REM CLEAR SCREEN
220 PRINT:PRINTTAB(13);"KNIGHT'S TOUR":PRINT
230 PRINT" -    THIS PROGRAM DISPLAYS A KNIGHT'S"
240 PRINT"  TOUR, THAT IS, A KNIGHT IN THE GAME"
250 PRINT"  OF CHESS CAN JUMP ON ALL 64 SQUARES"
260 PRINT"  OF A CHESS BOARD WITHOUT LANDING ON"
270 PRINT"  ANY SQUARE MORE THAN ONCE."
280 PRINT
290 OPEN 1,0
300 PRINT"  ENTER KNIGHT'S STARTING POSITION.":PRINT
310 PRINT"  ENTER THE ROW: ";:INPUT#1,R:PRINT
320 PRINT"  ENTER THE COLUMN: ";:INPUT#1,C:PRINT
330 CLOSE 1
340 IFR<1ORR>8ORC<1ORC>8THEN290
350 RETURN
360 :::::::::::::::::::::::::::::::::::::
400 REM     *** DEFINE GRAPHICS ***
410 PRINTCHR$(147);:REM CLEAR SCREEN
420 FORI=0TO63:READJ:POKE832+I,J:NEXTI
430 DATA,,,,,,,,,,,
440 DATA,8,,,30,,,63,,,127,
450 DATA,111,,,15,,,31,,,62,
460 DATA,62,,,28,,,62,,,127,
470 DATA,127,,,,,,,,,,,,,,
480 S=53248
490 POKES+21,1:POKES+39,1
500 POKES+23,1:POKES+29,1:POKE2040,13:POKE53281,14
510 P=1016:L=-1:FORJ=1TO8:FORM=1TO3
520 P=P+7:FORK=1TO8
530 FORN=1TO4:P=P+1:POKEP,160:POKEP+54272,ABS(L)*3
540 NEXTN:L=NOTL:NEXTK:P=P+1:NEXTM
550 L=NOTL:NEXTJ:RETURN
560 :::::::::::::::::::::::::::::::::::::
600 REM    *** DISPLAY SOLUTION ***
601 DATA3,42,5,20,37,40,15,18
602 DATA6,21,2,41,16,19,36,39
603 DATA43,4,57,54,59,38,17,14
```

```
604 DATA22,7,62,1,56,53,60,35
605 DATA49,44,55,58,61,64,13,28
606 DATA8,23,48,63,52,29,34,31
607 DATA45,50,25,10,47,32,27,12
608 DATA24,9,46,51,26,11,30,33
610 DIM BD(8,8),MD(8,8)
615 FORJ=1TO8:FORK=1TO8:READMD(J,K):NEXTK,J
620 N=0:X1=0:Y1=0
625 FC=MD(R,C)-1
626 FORJ=1TO8:FORK=1TO8:BD(J,K)=MD(J,K)-FC:IFBD(J,K)<1THENBD
    (J,K)=BD(J,K)+64
627 NEXTK,J
630 N=N+1:FORJ=1TO8:FORK=1TO8:IFBD(J,K)=NTHENR=J:C=K
633 NEXTK,J
635 P=1065+4*(C-1)+120*(R-1)
640 V=48+INT(N/10):POKEP,V
650 V=48+N-10*INT(N/10):POKEP+1,V
660 X2=32*C-18:Y2=24*R+16:DX=X2-X1:DY=Y2-Y1
670 IFABS(DY)>ABS(DX)THEN850
680 FORJ=SGN(DX)TODXSTEPSGN(DX)
690 X2=X1+J:Y2=Y1+J*DY/DX
700 POKES,X2:POKES+1,Y2
710 NEXTJ:IFN=64THENRETURN
750 X1=X2:Y1=Y2:GOTO630
850 FORJ=SGN(DY)TODYSTEPSGN(DY):Y2=Y1+J:X2=X1+J*DX/DY:GOTO700
860 ::::::::::::::::::::::::::::::::::::
```

	line 400 to define the graphics used in the program. Line 100 calls a graphic display of the solution. Line 110 allows you to clear the entire screen before ending the program. OPEN 1,0 allows the computer to read the keyboard for characters to be entered. INPUT#1,N$ is a command to input a character string without a question mark appearing. The POKE commands gets rid of the knight.
120	This is a line separating the main program from the first subroutine.
200-350	This subroutine prints the instructions and requests the starting square for the knight's tour. Notice the use of OPEN 1,0 to read the keyboard.
360	A separation line.
400	This line identifies the subroutine.
410	This line clears the screen.
420	This line reads the data from DATA statements in lines 430-470. These data are used to create four sprites that will make up a small knight on a chessboard. We read a total of 64 values.

430	This line consists of 12 zeros. Notice that they are not displayed but are separated by commas. Because 0 is nothing, it doesn't have to be displayed in BASIC.
440	This line consists of 12 values, four of which are nonzero.
450-470	These lines are the rest of the values used in defining the sprites for the knight.
480	S is the X position register of sprite 0.
490-500	These lines turn on the sprites. Line 500 also sets the background color to light blue and sets sprite 0's pointer.
510-550	These lines print out the chessboard on the screen. Each square is made of 3×4 solid color characters. Note the use of L=NOT L in lines 540 and 550. L alternates between −1 and 0 and is used to print the alternating square colors.
560	This separates two subroutine lines.
600	This is the start of the display subroutine.
601-608	This is a solution for the knight's tour if the knight starts at column 4, row 4 (left to right, top to bottom).
610	This line dimensions arrays for storing board values.
615	This line reads in the data of the original board from lines 601-608.
620-627	These lines rearrange the board data for the starting row and column. Observant programmers will notice that I cheated: I made the knight's tour circular. Then all I had to do was to rearrange the move numbers. In other words, you can reach the first move from the last move.
630-633	These lines find the position of the next move number.
635-650	These lines POKE the move number on the appropriate square on the screen.
660-850	These lines draw a straight line from the current knight position to the next square. It then moves the sprites along that line, one pixel at a time. This makes the knight appear to move smoothly along. After the 64th move, control goes to line 110.
860	This marks the end of the listing.

Program Operation. When you enter the program into RAM and type RUN, instructions are printed and you are asked for the row and column numbers of the knight's starting position. Enter an integer from 1 to 8 for the row and column. The computer will check for values outside of this range.

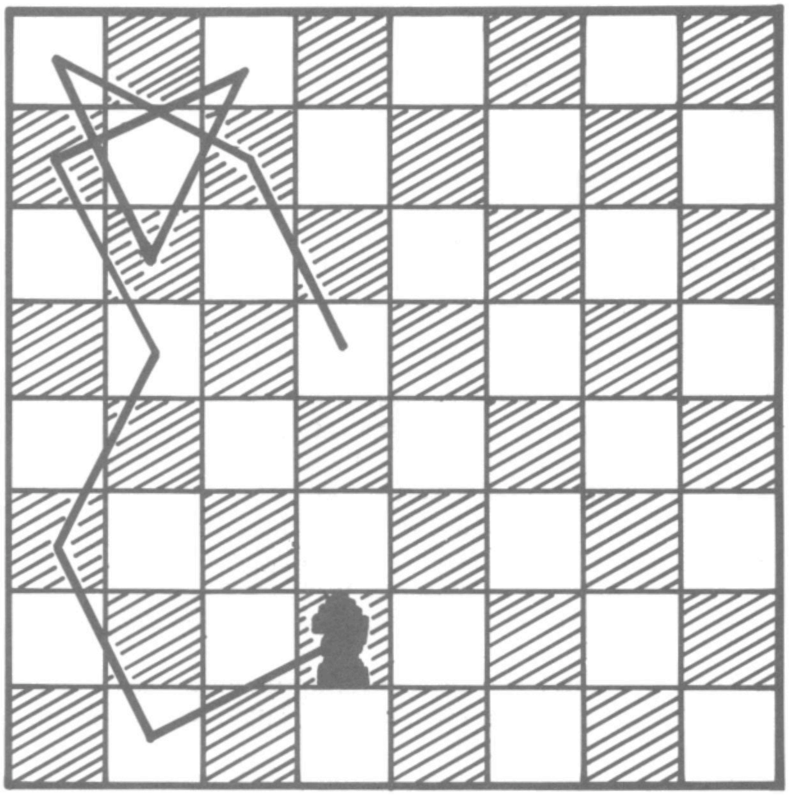


Fig. 1-4. Move 10 of Knight's Tour starting at Row 4, Column 4.

After that, the computer will print out a chessboard on the screen and change the screen color. A knight will appear from the upper left corner and will move around the board. The move numbers will be printed on the squares of the board just before the knight moves to the next square. These numbers act as a trace to leave a trail of the knight's tour. The graphics are interesting to follow.

You might want to think about how to change the program to have the computer solve other possible tours from the same initial starting position. The reason this program is included is that the computer uses one set of data to solve a wide range of possibilities. I think that makes it somewhat intelligent. Intelligence is really making the best use of stored information.

Chapter 2

Intelligent Games

In this chapter I discuss some intelligent computer games. The three games, the game of Nim, a maze program, and a cellular automaton program, show some differing aspects of artificial intelligence. I also discuss some of the traditional board games.

ARTIFICIAL INTELLIGENCE IN GAMES

Artificial intelligence is sometimes best understood by studying games. Because games are understood by everyone, even children, they are attractive candidates for the study of artificial intelligence on computers.

Artificial intelligence has not been fully utilized in computer games. Obviously as the technology grows, the capabilities of computers involved in computer games will similarly increase. Microcomputers are becoming faster, and more RAM is becoming available for the storage and execution of these games. Low-cost mass storage devices add to the capabilities of the microcomputer. These developments increase the possibilities for the future development of artificial intelligence in computer games.

Board Games

Four popular board games that have been computerized are chess, checkers, Othello, and backgammon. Let's discuss each one separately.

Chess. Chess is the most famous computer board game. There are microcomputer tournaments in which computers using different programs compete against one another. Large computers were first used to play the game of chess. It was then thought that the computer could not beat a grand master and that the use of computers in the game of chess was nothing more than an interesting experiment. People were startled when a computer first beat its human opponent at chess. Suddenly computers had to be taken seriously. There are now many chess programs for many kinds of microcomputers. Many are written in machine language to make maximum use of operating speed of the microprocessor.

In the future, chess programs will no doubt be written that play chess more like the way humans play the game. Typically the chess games in existence utilize tree searches, as mentioned earlier. Humans usually play chess by establishing a set of subgoals and then seeking the moves that will accomplish those subgoals. Many of the chess programs use the *brute force* method of deciding moves. A computer that would accept advice and make analogies between similar situations that it has "recognized" before would be operating in the realm of artificial intelligence. Chess is a rather complex game, so such a program would require considerable thought in programming. Certainly it shouldn't be written in BASIC.

Checkers. Checkers is another favorite computer board game. Again there are many strategies used in determining the moves made by the computer. Tree searches find the best possibilities. These searches don't give the program intelligence, however. If the computer could be programmed to remember a stategy from game to game, then it would be intelligent. A program would have to be designed so that it could change its own programming with time—or perhaps data files could be updated and read into the computer at the start of each game or series of games. This data file could then be saved at the end of the games, having been modified by "learned" strategy.

I don't think that checkers has a simple solution. If a relatively simple solution to the game could be found, the computer program could become an expert at the game. That is, the computer could find a mathematical solution for any given set of moves or states of the game.

Realistically, it might be reasonable for a computer to *learn* strategy. Humans look at a set of circumstances and generalize. They make a rule or set of rules that explains how the system works. They test that hypothesis against the situation. Where they are wrong they alter their thinking. This process can be defined as learning. A checkers program might be able to do similar things in the game of checkers.

Othello. There are several games of Othello in existence. In

Othello, pieces are won by "trapping" the opponent's pieces between one of your existing pieces and a piece you put down on a 8x8 board (see Fig. 2-1). There might be a binary solution for the game: discovering it would be an interesting project. There is at least one official Othello tournament for microcomputers.

Backgammon. Backgammon is another game that has been adapted for the computer. The simulated roll of the dice has been replaced by a random number generator. In July 1979 a computer backgammon program called BKG 9.8, written by Han Berlinger, defeated a world champion, Luisi Villa of Italy. Although this game was run on a large computer instead of a microcomputer, it represented the first time that a computer defeated a world champion at any board or card game. That program utilized artificial intelligence. (See the June, 1980 issue of *Scientific American,* Volume 242, Number 6, pages 64-72.) That program used several "intelligent" techniques to defeat its opponent, not

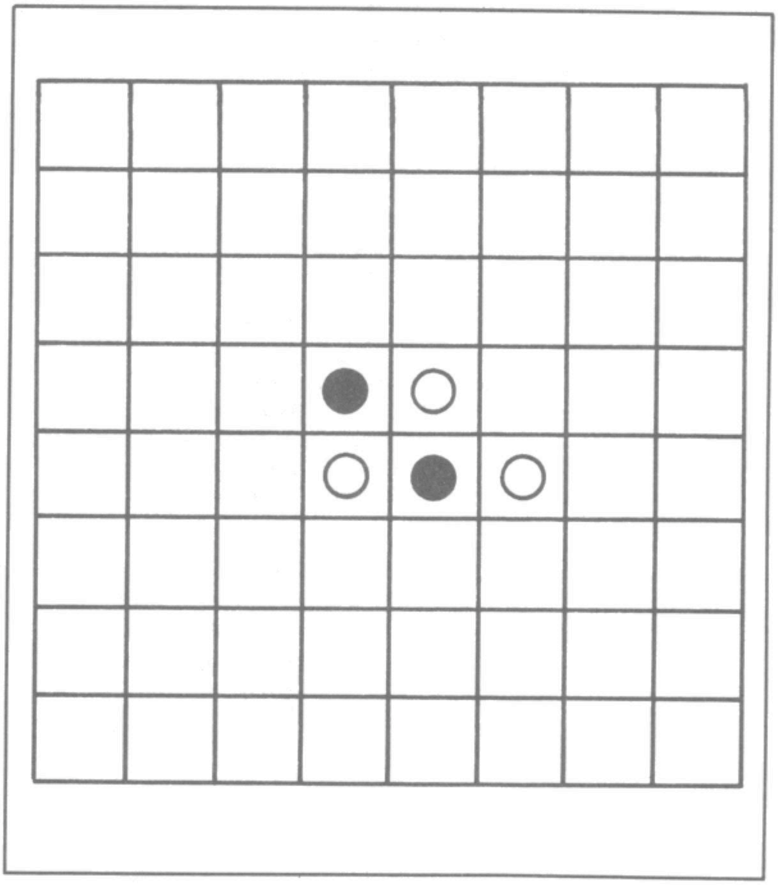


Fig. 2-1. An Othello board: the black piece has been "trapped" by the white pieces.

just brute force and mathematical calculation. It simulated some of the techniques that people use in playing the game.

THREE INTELLIGENT GAMES IN BASIC

Let's now look at the three games written in BASIC. The first is the game of Nim. The object of Nim is to pick from one to three items from a pile, leaving your opponent with the last item in the last pile. The second game is a maze program that the computer solves. The method involved is to successively change all three-sided squares (dead ends) to four-sided squares, effectively filling in the dead ends. The final result is the solution. The last of the three programs is a cellular automaton program with an interesting twist.

The Game of Nim

The program in Listing 2-1 plays the game of Nim with you. You have to match wits against the computer. The program is artificially intelligent because it will learn your winning strategy! Your moves are stored in an array that can be saved on tape at the end of the game or the series of games. (Listing 2-2 shows one way

Listing 2-1 The Game of NIM

```
10 :::::::::::::::::::::::::::::::::::::
15 REM     GAME OF NIM - PROGRAM 4
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
25 REM COPYRIGHT 1984, TAB BOOKS INC.
30 REM (WRITTEN FOR THE COMMODORE 64)
35 :::::::::::::::::::::::::::::::::::::
37 REM       *** MAIN PROGRAM ***
40 GOSUB 105:REM INITIALIZE PROGRAM
45 IF F=0 THEN GOSUB 710:GOSUB 405:GOSUB 540:IF TL=0 THEN 70
50 GOSUB 710:GOSUB 200:GOSUB 540:IF TL=0 THEN 70
55 IF F=1 THEN GOSUB 710:GOSUB 405:GOSUB 540:IF TL=0 THEN 70
60 GOTO 45
70 INPUT"WANT TO PLAY AGAIN (Y/N)";A$
80 PRINT:IF LEFT$(A$+" ",1)="Y" THEN GOSUB 119:GOTO 45
90 INPUT"WANT TO SAVE STRATEGY (Y/N)";A$
92 PRINT:IF LEFT$(A$+" ",1)="N" THEN END
94 OPEN 1,1,1,"STRATEGY":FOR I=1 TO 512
96 PRINT#1,P%(I):NEXT:CLOSE 1:END
98 :::::::::::::::::::::::::::::::::::::
100 REM   *** INITIALIZE PROGRAM ***
105 DIM P%(584),Q%(3,24)
110 GOSUB 810
114 A$="Y":INPUT"ENTER STRATEGY FROM TAPE (Y/N)";A$
115 PRINT:IF LEFT$(A$+" ",1)="N" THEN 119
116 OPEN 1,1,0,"STRATEGY":FOR I=1 TO 584
117 INPUT#1,P%(I):IF ST=0 THEN NEXT
118 CLOSE 1
```

```
119 M=1:F=0
120 IF RND(1)<0.5 THEN PRINT "YOU MAY MOVE FIRST.":F=1:GOTO 140
130 PRINT "THE COMPUTER WILL MOVE FIRST."
140 FOR I=1 TO 3
150 Q%(I,1)=3+5*RND(1)
160 FOR J=2 TO 24
170 Q%(I,J)=0
180 NEXT:NEXT
190 GOSUB 460:RETURN
195 ::::::::::::::::::::::::::::::::::::::
200 REM       *** YOUR MOVE ***
210 OPEN 1,0:PRINT "YOUR MOVE. ":PRINT
220 PRINT "ENTER PILE #:";:INPUT#1,PN:PRINT
230 IF PN<1 OR PN>3 THEN PRINT "ILLEGAL PILE #. TRY AGAIN.":PRINT:GOTO
    220
235 IF Q%(PN,M)=0 THEN PRINT "NONE IN THAT PILE. TRY AGAIN.":PRINT:
    GOTO 220
240 PRINT:PRINT "ENTER NUMBER TAKEN FROM PILE #"PN":";:INPUT#1,NB:
    PRINT
250 IF NB>3 THEN PRINT "TOO MANY. 3 IS MAXIMUM. TRY AGAIN.":PRINT:GOTO
    240
260 IF NB<1 THEN PRINT "COME ON NOW. TAKE AT LEAST 1.":PRINT:GOTO 240
270 IF Q%(PN,M)<NB THEN PRINT "ILLEGAL MOVE. TRY AGAIN.":PRINT:GOTO
    220
280 CLOSE 1:PRINT:PRINT "VERY WELL.":PRINT
290 Q%(0,M)=(PN-1)*3+NB
300 FORI=1TO3
310 Q%(I,M+1)=Q%(I,M)+NB*(I=PN)
320 NEXT:M=M+1:RETURN
330 ::::::::::::::::::::::::::::::::::::::
400 REM     *** COMPUTER'S MOVE ***
405 PRINT "COMPUTER'S MOVE.":PRINT
410 C=Q%(1,M)+Q%(2,M)*8+Q%(3,M)*64
420 IF P%(C)=0THEN GOSUB 480:GOTO 450
430 PN=1-(P%(C)>3)-(P%(C)>6)
440 NB=P%(C)-(PN-1)*3
450 PRINT "COMPUTER WILL TAKE"NB"FROM PILE #"PN".":PRINT
455 GOSUB 290
460 TI$="000000"
465 IF TI$<"000004" THEN 465
466 RETURN
470 ::::::::::::::::::::::::::::::::::::::
475 REM  *** RANDOM COMPUTER MOVE ***
480 PN=1+INT(3*RND(1))
490 IFQ%(PN,M)=0THENPN=PN+1:PN=PN+3*(PN>3):GOTO490
500 J=Q%(PN,M):NB=1+INT(-RND(1)*(J>1)-RND(1)*(J>2))
510 RETURN
520 ::::::::::::::::::::::::::::::::::::::
530 REM       *** IS GAME OVER? ***
540 TL=0:FOR I=1 TO 3
550 TL=TL+Q%(I,M)
560 NEXT:IF TL THEN RETURN
570 PRINT CHR$(147);"                GAME OVER!":PRINT
```

```
580 IF INT((M-F)/2)=(M-F)/2 THEN PRINT"COMPUTER LOST! YOU WON!":GOTO
    600
590 PRINT "YOU LOST! COMPUTER WON!"
600 PRINT:FORI=M-2 TO 1 STEP -2
605 PN=1-(Q%(0,I)>3)-(Q%(0,I)>6)
606 NB=Q%(0,I)-(PN-1)*3
610 P%(Q%(1,I)+Q%(2,I)*8+Q%(3,I)*64)=Q%(0,I)
611 IF PN=1 THEN P%(Q%(1,I)+Q%(3,I)*8+Q%(2,I)*64)=Q%(0,I)
612 IF PN=2 THEN P%(Q%(3,I)+Q%(2,I)*8+Q%(1,I)*64)=Q%(0,I)
613 IF PN=3 THEN P%(Q%(2,I)+Q%(1,I)*8+Q%(3,I)*64)=Q%(0,I)
614 IF PN=1 THEN P%(Q%(2,I)+Q%(1,I)*8+Q%(3,I)*64)=3+NB
615 IF PN=1 THEN P%(Q%(3,I)+Q%(1,I)*8+Q%(2,I)*64)=3+NB
616 IF PN=2 THEN P%(Q%(1,I)+Q%(3,I)*8+Q%(2,I)*64)=6+NB
617 IF PN=2 THEN P%(Q%(3,I)+Q%(1,I)*8+Q%(2,I)*64)=6+NB
618 IF PN=3 THEN P%(Q%(3,I)+Q%(1,I)*8+Q%(2,I)*64)=NB
619 IF PN=3 THEN P%(Q%(3,I)+Q%(2,I)*8+Q%(1,I)*64)=NB
620 NEXT:RETURN
630 :::::::::::::::::::::::::::::::::::::
700 REM       *** PRINT PILES ***
710 PRINT CHR$(147):FOR I=1 TO 3
720 PRINT "PILE #"I":   ";
730 FORJ=1 TO Q%(I,M)
740 IF J>Q%(I,M) THEN 780
750 PRINT CHR$(111)CHR$(17)CHR$(157);
760 PRINT CHR$(108)CHR$(186)CHR$(145);
770 PRINT CHR$(157)CHR$(112)" ";
780 NEXT:PRINT:PRINT:PRINT:NEXT:RETURN
790 :::::::::::::::::::::::::::::::::::::
800 REM       *** INSTRUCTIONS ***
810 PRINT CHR$(147);"            *** GAME OF NIM ***":PRINT:PRINT
820 PRINT"       IN THIS GAME YOU PICK FROM 1 TO 3"
830 PRINT"OBJECTS FROM ANY SINGLE PILE. EACH OF"
840 PRINT"THREE PILES WILL CONTAIN FROM 3 TO 8"
850 PRINT"OBJECTS. YOU WIN BY MAKING THE COMPUTER"
860 PRINT"PICK UP THE LAST OBJECT.":PRINT:PRINT
870 RETURN
880 :::::::::::::::::::::::::::::::::::::
```

Listing 2-2 Changes To Use Nim on Disk

```
70 INPUT"WANT TO PLAY AGAIN (Y/N)";A$
80 PRINT:IF LEFT$(A$+" ",1)="Y" THEN GOSUB 119:GOTO 45
85 INPUT"WANT TO SAVE STRATEGY (Y/N)";A$
87 PRINT:IF LEFT$(A$+" ",1)="N" THEN END
90 PRINT"IF YOU HAVE USED THIS PROGRAM BEFORE AND HAVE ALREADY SAVED
   A FILE ";
91 PRINT"OF STRATEGY  ON THIS DISK, ";
92 PRINT"IT WILL BE REPLACED BY"
93 PRINT"THE CURRENT FILE. ENTER Y IF SUCH A FILE ALREADY EXISTS."
94 INPUT NZ$:IF NZ$="Y" THEN OPEN 15,8,15:PRINT#15,"S0:STRATEGY":
   CLOSE15
95 OPEN 2,8,2,"STRATEGY,S,W":FOR I=1 TO 512
96 PRINT#2,P%(I):NEXT:CLOSE 2:END
```

```
98 :::::::::::::::::::::::::::::::::::::
100 REM    *** INITIALIZE PROGRAM ***
105 DIM P%(584),Q%(3,24)
110 GOSUB 810
114 A$="Y":INPUT"ENTER STRATEGY FROM DISK (Y/N)";A$
115 PRINT:IF LEFT$(A$+" ",1)="N" THEN 119
116 OPEN 2,8,2,"STRATEGY,S,R":FOR I=1 TO 584
117 INPUT#2,P%(I):IF ST=0 THEN NEXT
118 CLOSE 2
```

you can change the program so the strategy can be saved on disk. The other lines remain the same.) You can read that array into the program when you start playing the next time, and the computer should become more adept at playing the game. We might say that the computer is "learning" the game. It has gained experience.

The object of the game of Nim is to select objects from one of three piles. You must take one to three objects from only one pile. You win by making your opponent (the computer) pick up the last object. One kind of winning strategy is to leave two piles of two objects each, that would ensure a win for you because no matter which pile the computer picks from, you can pick from the other, leaving only one item. You will find other winning strategies—but so will the computer

We will now look at the game of Nim by examining the program line by line. In this game the computer will store winning moves as a code in an array. When it encounters those moves again, it will play those winning moves. If the moves are new to it, it will move randomly.

Here is an explanation of the lines of the game of Nim:

LINE	**EXPLANATION**
10-35	This is the title and credit information for the program.
37-98	This is the main program. Line 37 is a remark identifying the main part. Line 40 is a subroutine call to the beginning of the initialization subroutine. Line 45 says that if the computer moves first, it should go to the routine that prints the objects, then to the routine that computes the computer's move, and then to the routine that checks to see if the game is over. If it is, control goes to line 70. Line 50 calls the subroutine that displays the objects, the routine that lets you play your move, and the routine that checks to see if the game is over. If the game is over, then the computer goes to line 70. Line 55 is like line 45 except it is run if you moved first. Line 60 cycles control back to line 45.

Line 70 asks if you want to play another game. Line 80 checks your input to determine whether or not you want another game. If you do, part of the initialization subroutine is run (starting at line 119). Then control is returned to line 45. Line 90 asks if you want to save the computer's newest strategy on tape. Line 92 ends the program if you don't want to save the strategy. Lines 94-96 save the array, P%, on tape and ends the program.

100	This line is the title of the subroutine as a remark statement.
105	This line dimensions arrays P% and Q% stores the complete strategy of the game, Q% stores the moves of the current game being played.
110	This line calls the subroutine that prints the instructions.
114	This line asks if you want to enter strategy from earlier games from tape.
115	If the answer is negative, control jumps to line 119.
116-118	These lines properly read in the P% array from the cassette tape.
119	M is the move number. F is a flag indicating who moves first. If F=0, the computer moves first.
120	Based on a random number, you will move first, about half the time and F will equal 1. A jump is then made to line 140.
130	The computer will go first.
140-180	These lines blank the array, Q%, and fill in the number of objects in the three piles at the start of the game. The number of objects in each pile ranges from 3-7. The I index is the pile number, and the second index, J, is for the move number of this two-dimensional array.
190	This subroutine call is for a four second time delay starting at line 460.
200-320	This subroutine allows you to enter your move. It checks to see if you entered an incorrect number. If so, it informs you and you have to try again. Line 210 opens a keyboard file, for checking the characters that are being pressed. This is an alternative to using an INPUT statement with its question mark. PN is the pile number, and NB is the number of objects taken from pile PN. Line 290 calculates a code based on the pile number and the number taken. It is stored in array Q%. Lines 300-320 subtracts the number taken from the pile and copies the other piles in Q% for the next move. Note the use of the

	logical conditions, I = PN to compute the number in each pile for the next move. After that the move number is increased and control is returned to the main program.
400-466	This is the subroutine that computes the computer's best move. Line 410 computes a code from the number of objects in each pile to "look up" in array P%. The current number of objects in pile 1 is added to 8 times the number of objects in pile 2, and the result is added to 64 times the number of objects in pile 3. Line 420 says that if the corresponding lookup value is 0, then go to the subroutine at line 480 to make a random move, and then go to line 450. Line 430 computes the pile number to pick from. Notice the use of logical conditions in computing the value. Line 440 computes the number of objects to take from the pile. Line 450 prints out how many objects it will take and from which pile. 455 is a call to line 290, part of the "your move" subroutine. This call updates the Q% array and increases the move number as before. After that, control is returned to line 460, which sets a timer at zero. This timer gives you four seconds to read what the computer just printed on the screen. Line 465 actually uses the timer. If you want to change the time delay, increase or decrease the numbers in the quotes. Line 466 returns control to the main program.
470	This line is a subroutine separator.
475-510	This subroutine chooses a random number of objects to be taken from a random pile containing objects.
530-620	This subroutine checks to see if the game is over. If it is, then P% is updated with the winning moves. Lines 540-560 check to see if any objects are left in the three piles. Lines 600-620 update the P% array with the winning moves. Then control is returned to the main program.
700-780	These lines print the graphics of the game. The objects appear as squares. Code 111 is the upper left corner, 17 is the code for cursor down, 157 is the bottom left corner, and so forth.
800-880	These lines clear the screen and print the instructions at the beginning of the game.

Program Operation. After you enter the program in memory, and type RUN, the program will display the instructions. You are then asked if you want to enter strategy from tape. If you had saved strategy on tape from when you had played the game before, type Y.

If not, type N. The computer will then determine who moves first. It will then make three piles of objects with three to seven items in each. These will be displayed on the screen as squares. You are asked to enter the pile number that you wish to pick from; then you are asked for the number of objects that you wish to pick. If you make a mistake, the computer will inform you, and you will have to reenter the move. You and the computer will take turns moving.

After the game is over, you will be asked if you want to play again. Type Y for Yes or N for No. If you type Y, you will play another game. If you type N, the computer will ask you if you want to save the current level of strategy "learned" by the computer on tape. If you want to, type Y for Yes, or type N for No. After that the program will end. You will notice that the computer will gradually make some of the same moves that you made to attempt to win games. The computer will move randomly when it has no recorded strategy for a given move. After each game, the computer will convert the moves to a code and store those codes in the array P%. When you save the strategy on tape, you are actually storing the numbers in this array.

This program is a good example of artificial intelligence in games because the computer's performance improves with the number of games played, something that we might consider to be learning. It is a good idea to save strategy on tape after every series of games. You might want to convert the programs to save more than one file of strategies on disk if you have a disk drive. You would have to change the OPEN statements to the appropriaté disk drive file numbers and device numbers.

The Path Through a Maze

The Maze program in Listing 2-3 shows how the computer can find a path through a maze. It uses a technique of changing cells, or parts of a maze, to find the solution to the maze. By debranching dead ends in the maze, the computer successfully approaches the solution to the puzzle. After no more cells can be changed, the computer prints a solid square in all of the cells that have not been changed. The program assumes that there are an entrance and an exit to the maze.

Listing 2-3 The Path through a Maze

```
10 ::::::::::::::::::::::::::::::::::::::
20 REM          MAZE - PROGRAM 5
30 REM WRITTEN BY TIMOTHY J. O'MALLEY
40 REM COPYRIGHT 1983, TAB BOOKS INC.
50 REM (WRITTEN FOR THE COMMODORE 64)
60 ::::::::::::::::::::::::::::::::::::::
70 REM          *** MAIN PROGRAM ***
80 GOSUB 200:REM INSTRUCTIONS
100 GOSUB 600:REM READ MAZE DATA
```

```
110 GOSUB 720:REM DETERMINE SOLUTION
120 OPEN 1,0:INPUT#1,AI$:PRINT"{HOME}":END
130 ::::::::::::::::::::::::::::::::::::
200 REM      *** INSTRUCTIONS ***
205 PRINT"{WHT}";
207 POKE 53281,4
210 PRINTCHR$(147);TAB(12);"MAZE - PROGRAM 5":PRINT:PRINT
220 PRINT"     THIS PROGRAM ALLOWS THE COMPUTER TO"
230 PRINT"FIND ITS WAY THROUGH A MAZE. THE MAZE "
240 PRINT"IS DEFINED IN THE DATA STATEMENTS IN "
250 PRINT"THE PROGRAM LISTING, LINES 640-710."
260 PRINT:INPUT"PRESS A KEY TO CONTINUE. OK";AI$
390 RETURN
395 ::::::::::::::::::::::::::::::::::::
400 REM *** CODES FOR SIDES OF MAZE ***
401 PRINT"▔▔▔▔{C/LF}{C/LF}{C/LF}{C/LF}";:RETURN
402 PRINT"{C/RT}{C/RT}{C/RT} ▌{C/LF}{C/DN} ▌{C/LF}{C/DN} ▌{C/LF}{C/LF}
   {C/LF}{C/LF} {C/UP}{C/UP}";:RETURN
403 PRINT"{C/RT}{C/RT}{C/RT}┐{C/LF}{C/LF}{C/LF}{C/LF}";:RETURN
404 PRINT"{C/DN}{C/DN}▁▁▁▁{C/LF}{C/LF}{C/LF}{C/LF}{C/UP}{C/UP}";:
    RETURN
406 PRINT"{C/DN}{C/DN}{C/RT}{C/RT}{C/RT}┘{C/LF}{C/LF}{C/LF}{C/LF}
    {C/UP}{C/UP}";:RETURN
408 PRINT"▌ {C/LF}{C/DN}▌ {C/LF}{C/DN}▌ {C/LF}{C/UP}{C/UP}";:RETURN
409 PRINT"┌{C/LF}";:RETURN
412 PRINT"{C/DN}{C/DN}└{C/LF}{C/UP}{C/UP}";:RETURN
590 ::::::::::::::::::::::::::::::::::::
600 REM      *** READ MAZE DATA ***
610 DIM B(8,8)
615 PRINT "{CLR}";
620 FORI=1 TO 8:FOR J=1 TO 8
622 READ B(I,J):IF B(I,J)=0 THEN 639
624 FOR K=0 TO 3:ON ((B(I,J) AND 2↑K)=2↑K)*-(K+1) GOSUB 401,402,404,
    408
625 NEXT:FOR K=1 TO 4
630 ON ((B(I,J) AND 3*K)=3*K)*-K GOSUB 403,406,409,412
635 NEXT
639 PRINT"{C/RT}{C/RT}{C/RT}{C/RT}";:NEXT:PRINT:PRINT:PRINT:NEXT
640 DATA 10,9,3,9,5,3,13,3
650 DATA 10,14,10,10,13,4,5,2
660 DATA 12,5,4,6,9,5,3,10
670 DATA 11,9,5,5,6,11,10,10
680 DATA 10,10,13,5,1,6,12,6
690 DATA 10,12,5,5,0,5,5,7
700 DATA 8,5,5,7,10,9,5,3
710 DATA 12,5,5,5,4,4,7,10
715 RETURN
717 ::::::::::::::::::::::::::::::::::::
720 PRINT "{HOME}";:F=0:FOR I=1 TO 8:FOR J=1 TO 8
730 IF B(I,J)=7 THEN F=1:B(I,J)=15:B(I,J-1)=B(I,J-1)+2
740 IF B(I,J)=11 THEN F=1:B(I,J)=15:B(I+1,J)=B(I+1,J)+1
750 IFB(I,J)=13 THEN F=1:B(I,J)=15:B(I,J+1)=B(I,J+1)+8
760 IF B(I,J)=14 THEN F=1:B(I,J)=15:B(I-1,J)=B(I-1,J)+4
```

```
770 NEXT J,I
780 IF F THEN 720
790 FOR I=1 TO 8:FOR J=1 TO 8
800 IF B(I,J)=15 THEN PRINT "{C/RT}{C/RT}{C/RT}{C/RT}";:GOTO 810
806 PRINT "{C/DN}{C/RT} ▮▮{C/RT}{C/UP}";
810 NEXT:PRINT:PRINT:PRINT:NEXT:RETURN
820 :::::::::::::::::::::::::::::::::::
```

This program represents an unusual method of solving the maze problem. Typically, methods have utilized a stack array in which the computer would store. Whenever the computer would reach a dead end, the computer would pop moves off of the stack until it reached a branch point. Then it would continue putting moves on the stack. The method used in this program is a faster way to solve the problem.

Here is a line by line description of the program and its operation.

LINE	**EXPLANATION**
10-60	These lines identify the program.
70-120	This is the main program. It is a series of subroutine calls that solves the maze puzzle. Line 70 identifies the main program. Line 80 is a call to a subroutine that prints out the instructions. Line 100 is a call to a routine that reads the maze data. Line 110 is a subroutine call to line 720 where the solution is determined. Line 120 waits for you to press the RETURN key after the program is over. It then ends the program.
130	This line separates the parts of the program.
200-390	This set of lines prints the instructions. Line 205 is a command that changes the color of the characters to white. {WHT}is produced by pressing the CTRL and 2 keys simultaneously. This character does not appear this way on the screen but is the way some printers print it. Line 207 is a POKE command that changes the screen colors.
400-412	These lines are a set of subroutines that print the sides of the cells in the maze. Line 401 contains four graphic characters (the left one on the Y key). {C/LF} is the CRSR LEFT key. Line 402 contains the CRSR RIGHT characters, {C/RT}, followed by the left graphic character on the N key. The CRSR DOWN {C/DN} characters and the CRSR UP {C/UP} control characters are also used. Line 403 contains some of these same control characters plus the right graphic character on the P key. Line 404 contains

control characters plus the left graphic character on the P key. line 406 contains control characters plus the right graphic character on the @ key. Line 408 contains control characters plus the left graphic character on the H key. Line 409 contains the right graphic character on the O key. Line 412 contains the right graphic character on the L key plus control characters. Together these subroutines draw parts of the maze.

600–715 These lines dimension the B array, clear the screen, and read in the elements of the B array. Line 615 contains the {CLR} character produced by pressing the SHIFT and CLR/HOME keys. The maze is made up of eight rows and eight columns of cells. The numbers of those cells are contained in lines 640–710. The cells' numbers are first defined as zero. If the cell has a line on top, a one is added to its number. If the cell has a line on the right, a two is added to the cell's number. If the cell has a line on the bottom, a four is added to the cell's number. If the cell has a line on the left, an eight is added to the cell's number. A cell with a number of 10 has a line on the left and a line on the right. The top and bottom are open. A cell with a value of 9 has a top and bottom but no sides. That is how the numbers on the DATA statements in lines 640–710 make up the maze. You might want to change these values or have the computer change these numbers and then solve the maze puzzle. The rest of the lines of this subroutine draw the lines of the maze, based on the cells' values.

720–810 This is the subroutine that actually solves the maze problem. Line 720 prints the HOME character, sets a F flag to 0, and begins two loops that are nested. These loops look for cells that have 1 open side. Line 730 converts a cell with an opening on the left to a solid square, effectively debranching that dead end. It also sets the F to 1, indicating that an operation has taken place. It also changes the cell to the left by adding a right side to it. Line 740 does similar things to a cell with no bottom. It also adds a top to the cell beneath it and sets the F flag variable. Line 750 adds a right side to the cell without one and adds a left side to the cell on its right. Line 760 adds a top line to a cell without one and adds a bottom line to the cell above it. This process is repeated over and over until no F flags are set. Then the program prints the left graphic character on the + key. This character is represented in in line 806 as a black square with a white column in it. On the screen it appears as a checkered square.

820 This indicates the end of the program.

Program Operation. When this program is correctly entered and run, it will print out the maze as an eight by eight collection of cells. The maze has an open end on the top and an open end on the bottom. The computer will solve the puzzle and then print a checkered square in all of the cells that were not debranched. This represents the solution to the puzzle. Just follow the squares from one end to the other. This trial of characters is the solution to the maze puzzle.

You might like to change the data in lines 640–710 to make a new maze. Or you might like to change the program so that the computer will generate a new maze every time that the program is run. You might also want to change the number of cells that are displayed on the screen.

The Cellular Automaton

This program, shown in Listing 2-4, needs a little explanation. First of all, you must enter the program into the Commodore 64 EXACTLY as it is listed. The reason is that the program will change the characters in the DATA statements as the program is run! That is, if you run the program and then press STOP and look at the program listing, it will appear different than before it was run! If you do not enter the program exactly, the program may crash. Figures 2-2 and 2-3 shows how the data statements change as the program runs.

Listing 2-4 Cellular Automaton

```
0 :::::::::::::::::::::::::::::::::::
1 REM CELLULAR AUTOMATON - PROGRAM 6
2 REM WRITTEN BY TIMOTHY J. O'MALLEY
3 REM COPYRIGHT 1983, TAB BOOKS INC.
4 REM (WRITTEN FOR THE COMMODORE 64)
5 :::::::::::::::::::::::::::::::::::
9 DIM B%(25,30),D%(25,30)
10 FOR I=1 TO 25:READ B$:FOR J=1 TO 30
20 IF MID$(B$,J,1)<>" " THEN B%(I,J)=1
30 IF MID$(B$,J,I)=" " THEN B%(I,J)=0
40 NEXT J,I
51 DATA"                              "
52 DATA"                              "
53 DATA"                              "
54 DATA"                              "
55 DATA"                              "
```

```
56 DATA"                              "
57 DATA"                              "
58 DATA"                              "
59 DATA"                              "
60 DATA"                              "
61 DATA"                              "
62 DATA"             ***              "
63 DATA"             * *              "
64 DATA"             * *              "
65 DATA"                              "
66 DATA"                              "
67 DATA"                              "
68 DATA"                              "
69 DATA"                              "
70 DATA"                              "
71 DATA"                              "
72 DATA"                              "
73 DATA"                              "
74 DATA"                              "
75 DATA"                              "
76 PRINT CHR$(147);
77 FOR I=55296 TO 56295:POKE I,1:NEXT
80 FOR I=1 TO 25:FOR J=1 TO 30:D%(I,J)=0
90 NEXT J,I:FOR I=2 TO 24:FOR J=2 TO 29
100 IF B%(I,J)=0 THEN 130
110 FOR K=I-1 TO I+1:FOR L=J-1 TO J+1
120 D%(K,L)=D%(K,L)+1:NEXT L,K:D%(I,J)=D%(I,J)-1
130 NEXT J,I:FOR I=2 TO 24:R=983+40*I:U=2413+38*I:FOR J=2 TO 29
140 T=U+J:P=R+J:B%(I,J)=-(D%(I,J)=3 OR (B%(I,J)=1 AND D%(I,J)=2))
150 S=32+10*B%(I,J):POKE P,S
160 POKE T-39,S
170 NEXT J,I:GOTO 80
180 :::::::::::::::::::::::::::::::::::
```

This program is a cellular automaton. That is, starting from an initial pattern, you can generate very sophisticated patterns. This program is based upon the game of life that was invented in 1970 by John Horton Conway. (See "Computer Recreations" in *Scientific American,* Volume 250, Number 3, pages 12–21 for a more detailed explanation on cellular automata.) This program stores the state of the automaton by changing the program.

In this program, the pattern of asterisks and blanks determine where the next generation of asterisks and blanks will appear. Here are the rules that control the changes. If a cell touches three other cells or if that cell is an asterisk and touches two other asterisks, it will be an asterisk in the next generation. All other conditions for cells will cause them to "die" and become blanks. This causes the patterns to change.

You can see that as the program runs, the patterns will change in unpredictable ways. A spinoff effect of this program is that the program itself will change because the DATA statements will

```
51 DATA"                            "
52 DATA"                            "
53 DATA"                            "
54 DATA"                            "
55 DATA"                            "
56 DATA"                            "
57 DATA"                            "
58 DATA"                            "
59 DATA"                            "
60 DATA"                            "
61 DATA"                            "
62 DATA"           ***              "
63 DATA"           * *              "
64 DATA"           * *              "
65 DATA"                            "
66 DATA"                            "
67 DATA"                            "
68 DATA"                            "
69 DATA"                            "
70 DATA"                            "
71 DATA"                            "
72 DATA"                            "
73 DATA"                            "
74 DATA"                            "
75 DATA"                            "
```

Fig. 2-2. The data statements in the Cellular Automaton program at the start.

change. This means that you can STOP the program at any time and SAVE it. When you restart that saved program, it will start working based on the newly stored pattern in the DATA statements.

Think of how other programs that would change the program lines as they ran might be written. This process can have very interesting consequences! Cellular automata should be explored further for their usefulness in computing. Someday when parallel processing becomes commonplace in microcomputers, cellular automata will reach their potential.

Here is a line-by-line description of the program.

LINE	EXPLANATION
0–5	Note the use of line 0 as part of the heading of this program. Zero is a valid line number that usually is not used in programs.

①

```
51 DATA"               "
52 DATA"               "
53 DATA"               "
54 DATA"               "
55 DATA"               "
56 DATA"               "
57 DATA"               "
58 DATA"               "
59 DATA"               "
60 DATA"               "
61 DATA"       *       "
62 DATA"      * *      "
63 DATA"     ** **     "
64 DATA"               "
65 DATA"               "
66 DATA"               "
67 DATA"               "
68 DATA"               "
69 DATA"               "
70 DATA"               "
71 DATA"               "
72 DATA"               "
73 DATA"               "
74 DATA"               "
75 DATA"               "
```

②

```
51 DATA"               "
52 DATA"               "
53 DATA"               "
54 DATA"               "
55 DATA"               "
56 DATA"               "
57 DATA"               "
58 DATA"               "
59 DATA"               "
60 DATA"               "
61 DATA"       *       "
62 DATA"     ** **     "
63 DATA"     ** **     "
64 DATA"               "
65 DATA"               "
66 DATA"               "
67 DATA"               "
68 DATA"               "
69 DATA"               "
70 DATA"               "
71 DATA"               "
72 DATA"               "
73 DATA"               "
74 DATA"               "
75 DATA"               "
```

③

```
51 DATA"               "
52 DATA"               "
53 DATA"               "
54 DATA"               "
55 DATA"               "
56 DATA"               "
57 DATA"               "
58 DATA"               "
59 DATA"               "
60 DATA"               "
61 DATA"      ***      "
62 DATA"     *   *     "
63 DATA"     ** **     "
64 DATA"               "
65 DATA"               "
66 DATA"               "
67 DATA"               "
68 DATA"               "
69 DATA"               "
70 DATA"               "
71 DATA"               "
72 DATA"               "
73 DATA"               "
74 DATA"               "
75 DATA"               "
```

④

```
51 DATA"               "
52 DATA"               "
53 DATA"               "
54 DATA"               "
55 DATA"               "
56 DATA"               "
57 DATA"               "
58 DATA"               "
59 DATA"               "
60 DATA"       *       "
61 DATA"      ***      "
62 DATA"     *   *     "
63 DATA"     ** **     "
64 DATA"               "
65 DATA"               "
66 DATA"               "
67 DATA"               "
68 DATA"               "
69 DATA"               "
70 DATA"               "
71 DATA"               "
72 DATA"               "
73 DATA"               "
74 DATA"               "
75 DATA"               "
```

Fig. 2-3. The Data Statements in the Cellular Automaton program as they continue to change.

9	This line dimensions two integer arrays. These arrays store 1 or 0, indicating that a character (like the asterisk) is present in the data or that a blank was present. D% is the *working* array, an array that is made based on what is contained in the B% array. When all the cells of the B% have been checked, D% is changed into B%, and D% is reset to zero.
10-40	These loops read in the characters in the character strings of the DATA statements in lines 51-75. If the characters in those lines are blanks, then the corresponding values in B% are 0. All other characters are represented as 1 in B%.
51-75	These lines contain the "screen" of the cellular automaton. You may change the blanks or asterisks to whatever printable character that you like. You may design different patterns. Just make sure that you maintain the same number of characters in each line and you maintain the same number of lines. There are 30 characters in each line.
76	This clears the screen.
77	This turns on all of the 1000 screen positions.
80-90	This resets all of the D% array and begins to work with the B% array.
100	If B% is 0, skip it.
110-120	This nested loop adds 1 to all the surrounding cells in the D% array for the occupied cell in B%.
130-170	These lines change the cells in B% to those in the completed D% array. They also change the blanks and asterisks in the DATA statements by using the POKE command. Notice the complicated use of Boolean logic to determine what cells are changed.
180	This is the last line of the program.

Program Operation. When you type RUN, the screen will blank. The pattern will start to print out on the screen. Because the program POKEs the asterisks upon the screen, the patterns will appear to change without the screen being blanked out. The program will continue to run indefinitely. To stop the program, you must press the RUN/STOP key. Check out the listing. You will see that it has changed, and you might want to save the revised program.

Chapter 3

Behaviors and Bootstraps

In this chapter we look at two novel ideas. One is the idea of having the computer learn behaviors. The second is the idea of programs that change themselves.

AN INTRODUCTION TO BEHAVIOR

Experimental behavioral psychology is the science that deals with the relationships between stimuli and overt behaviors. Typically a stimulus is a measured quantity and the resulting behavior is a quantitative response to that stimulus. The experimental variable is the amount of the stimuli that results in the subsequent behavior relative to a control. Behavioral psychology, then, makes correlations and defines causal relationships between the stimulus and the response.

I explored that idea that a simulation of behavior could be created on the computer. The first program in this chapter simulates the behavior of a rat in a Skinner box. Let's look at some definitions before we look at that program.

Definitions

In experimental behavioral psychology, as with any "true" science, everything has to be defined mathematically. If we say that something might exist, we call it a *hypothetical construct.* (A hypothetical construct is an entity that may or may not have essence.) For science to deal effectively with a hypothetical construct, the construct has to be presented in terms of an operational definition. We must find a way to measure it. Our stimulus may be a drug measured in milligrams, and the response may be a drop in blood pressure measured as millimeters of mercury. If we notice a positive relationship between the stimulus and the response, we say that a correlation exists. If we can manipulate the values of the stimulus and notice a consequential change in the values of the response, we have established a causal relationship. A control is an identical experiment whose experimental variable has not been manipulated but has typically been set at zero. A null hypothesis says that the amount of the stimulus has no effect on the response. We disprove the null hypothesis when we establish the causal relationship. This is the core of all experimental science.

B. F. Skinner and His Ideas

At the risk of being ridiculed by some of the members of the artificial intelligence community, let's spend a few moments discussing B. F. Skinner and some of his ideas about behavior.

One of the assumptions of behavior psychology is that if a stimulus caused the frequency of a behavior to increase, that stimulus was acting as positive reinforcement. Conversely, if the frequency of the response decreased, then the stimulus was acting as negative reinforcement. In experiments food might act as positive reinforcement, particularly if the experimental animal was on a deprivation schedule. A mild shock administered after a behavior might act as negative reinforcement. The schedule of reinforcement could be varied. A variable schedule of reinforcement leads animals to exhibit superstitious behavior.

COMPUTER BEHAVIOR

We don't normally think of computers as having behavior. They are programmed and that's that, they are ruthlessly consistent, they are idiots with lightning reflexes. On the other hand, computers can be programmed to do almost anything.

Is it possible to write programs that will exhibit behavior? To answer that question we have to devise a way to control the frequency of responses. That can be done by using an array of numbers that correspond to specific responses. On the Commodore 64 we will use such a frequency array to control the responses of the computer in a behavioral sort of way.

Training the Rat

In the program that we now discuss the object is to train a rat.

The computer will act as the rat and will display certain behaviors that the rat would exhibit. You will have five seconds to press a key to positively reinforce the last behavior displayed. If you do not press a key, that behavior will actually decrease in frequency (by 5%). When you press the key, you will increase the frequency of that response by 10%. The set of behaviors can be saved on tape and retrieved for use in the program the next time that the program is run.

The program shown in Listing 3-1 allows you to simulate some of the behaviors of a rat in a Skinner box. The computer will act as the rat, and you can reinforce some of his behaviors by pressing any key within five seconds of the displayed behavior. Up to 20 different behaviors will be displayed on the screen. Certain behaviors are linked together. You can reinforce the sequence of behaviors that occur. If you don't press any key, the selected behavior will decrease in frequency.

Listing 3-1 Training the Rat

```
10 :::::::::::::::::::::::::::::::::::::::
20 REM   TRAINING THE RAT - PROGRAM 7
30 REM WRITTEN BY TIMOTHY J. O'MALLEY
40 REM COPYRIGHT 1984, TAB BOOKS INC.
50 REM (WRITTEN FOR THE COMMODORE 64)
60 :::::::::::::::::::::::::::::::::::::::
70 REM    *** MAIN PROGRAM ***
80 DIM Y(79),P(20),F(79),S(20),A$(20)
90 FOR J=1 TO 79:READ Y(J):F(J)=1:NEXT
100 FOR J=1 TO 20:READ P(J):READ S(J):NEXT
110 FOR J=1 TO 20:READ A$(J):NEXT
120 GOSUB 200:REM PRINT INSTRUCTIONS
130 X=3
135 K=S(X)+INT(P(X)*RND(1))
136 TF=0:FORI=S(X)TOS(X)+P(X)-1
137 TF=TF+F(I):NEXT
140 IF (RND(1)*TF)<F(K) THEN PRINT:PRINTA$(Y(K)):GOTO 150
145 GOTO 135
150 TI$="000000":R$=""
152 GET R$:IF R$="" AND TI$<"000005" THEN 152
153 IF R$=CHR$(133) THEN 700
155 IF R$<>"" THEN F(K)=F(K)*1.1:X=Y(K):GOTO135
160 F(K)=.95*F(K):X=Y(K):GOTO135
199 :::::::::::::::::::::::::::::::::::::::
200 REM   *** PRINT INSTRUCTIONS ***
210 PRINTCHR$(147);TAB(12);"REINFORCEMENT":PRINT:PRINT
220 PRINT"       THIS PROGRAM SIMULATES A SKINNER "
230 PRINT"BOX. THE DESCRIPTION OF THE BEHAVIOR OF"
240 PRINT"THE RAT WILL BE DISPLAYED ON THE"
250 PRINT"SCREEN. BY PRESSING ANY KEY, YOU CAN"
260 PRINT"POSITIVELY REINFORCE THAT BEHAVIOR."
270 PRINT"POSITIVELY REINFORCED BEHAVIOR WILL "
271 PRINT"INCREASE IN FREQUENCY.":PRINT
```

```
272 PRINT"      PRESS F1 TO STOP OR SAVE BEHAVIOR"
273 PRINT"ON CASSETTE TAPE.":PRINT
281 INPUT"DO YOU WANT TO ENTER BEHAVIOR (Y/N)";BA$
282 IF LEFT$(BA$+"Y",1)<>"Y" THEN 286
283 OPEN 1,1,0,"BEHAVIOR":FOR I=1 TO 79
284 INPUT#1,F(I):IF ST=0 THEN NEXT
285 CLOSE 1:F2=1
286 PRINT"PRESS THE RETURN KEY TO CONTINUE. OK";:INPUT B$
287 PRINTCHR$(147)
290 PRINT"      THE RAT HAS JUST BEEN PLACED IN"
300 PRINT"THE SKINNER BOX.":PRINT:RETURN
310 ::::::::::::::::::::::::::::::::::::
400 REM           *** DATA ***
410 DATA 2,1,3,5,6,7,8,1,4,5,6,7,8,1,4,5,6,7,8,3,5
420 DATA 1,3,5,6,7,8,13,3,5,6,7,8,17,3,9,10,3,9,10
430 DATA 11,12,11,12,3,9,10,14,15,16,1,3,5,6,7,8,13
440 DATA 14,15,16,14,15,16,18,19,20,1,3,5,6,7,8,17
450 DATA 18,19,20,18,19,20,1,1,6,2,6,8,6,14,2,20,7
460 DATA 22,6,29,3,35,3,39,2,41,2,43,3,45,3,48,7,51
470 DATA 3,58,3,61,3,64,7,67,3,74,3,77
480 DATA RAT APPROACHES PELLET DISPENSER.
490 DATA RAT TURNS AWAY FROM DISPENSER.
500 DATA RAT STANDS ON ALL FOUR LEGS.
510 DATA RAT CONTINUES STANDING ON ALL FOURS.
520 DATA THE RAT JUMPS.
530 DATA RAT WALKS AROUND ON RIGHT SIDE OF BOX.
540 DATA RAT WALKS AROUND ON LEFT SIDE OF BOX.
550 DATA THE RAT LIES DOWN.
560 DATA IT CONTINUES LYING DOWN.
570 DATA IT GOES TO SLEEP.
580 DATA RAT CONTINUES SLEEPING.
590 DATA THE RAT WAKES UP.
600 DATA THE RAT FACES THE BAR PRESS ON HIND LEGS.
610 DATA RAT TURNS AWAY FROM BAR PRESS.
620 DATA NOW THE RAT IS ONLY TOUCHING THE BAR.
630 DATA THE RAT IS PRESSING THE BAR PRESS.
640 DATA RAT FACES PUSH ROD ON HIND LEGS.
650 DATA IT TURNS AWAY FROM PUSH ROD.
660 DATA RAT IS ONLY TOUCHING THE PUSH ROD.
670 DATA THE RAT IS PUSHING THE ROD.
680 ::::::::::::::::::::::::::::::::::::
700 REM  *** SAVE BEHAVIOR ON TAPE? ***
710 PRINTCHR$(147)
720 INPUT"WANT TO SAVE BEHAVIOR ON TAPE (Y/N)";BC$
730 IF LEFT$(BC$+"Y",1)<>"Y" THEN 760
740 OPEN 1,1,1,"BEHAVIOR":FOR I=1 TO 79
750 PRINT#1,F(I):NEXT:CLOSE 1
760 PRINT:INPUT"WANT TO STOP (Y/N)";BD$
770 IF LEFT$(BD$+"Y")<>"Y" THEN 135
790 ::::::::::::::::::::::::::::::::::::
```

There is a frequency array that stores the frequency of every behavior, F. After any given behavior, the computer selects one of several possible subsequent behaviors. If the frequency of the selected behavior is very small, the computer will reselect another behavior. In this way the frequency of the behavior, as contained in array F, controls the chance of its occurring. You positively reinforce a behavior by pressing a key. (Its frequency will increase 10% each time.) You can extinguish a behavior by not pressing any key. (Its frequency will decrease 5% in that case.) You might want to change the program to alter these numbers. You might also want to add additional behaviors. The program as it is shown in Listing 3-1 allows you to save the rat's behavior on tape. Listing 3-2 shows one way to change the program for use with a disk drive. (The rest of the lines remain the same.) You may want to change the program so that you can collect a number of behavior files on the same disk.

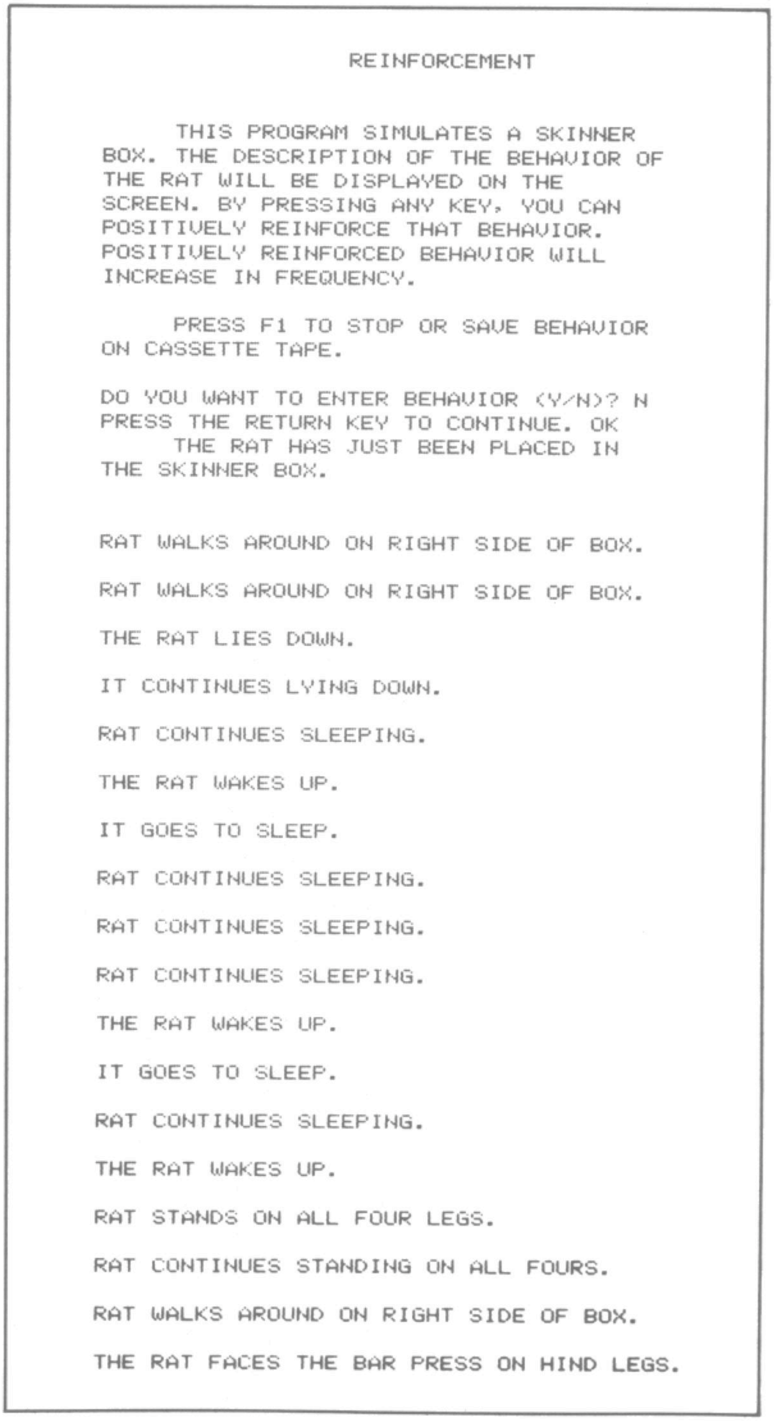
Listing 3-2 Changes To Use the Program on Disk

```
272 PRINT"      PRESS F1 TO STOP OR SAVE BEHAVIOR"
273 PRINT"ON DISK.":PRINT
281 INPUT"DO YOU WANT TO LOAD BEHAVIOR (Y/N)";BA$
282 IF LEFT$(BA$+"Y",1)<>"Y" THEN 286
283 OPEN 2,8,2,"BEHAVIOR,S,R":FOR I=1 TO 79
284 INPUT#2,F(I):IF ST=0 THEN NEXT
285 CLOSE 2:F2=1

700 REM  *** SAVE BEHAVIOR ON DISK? ***
710 PRINTCHR$(147)
720 INPUT"WANT TO SAVE BEHAVIOR ON DISK (Y/N)";BC$
730 IF LEFT$(BC$+"Y",1)<>"Y" THEN 760
732 PRINT"IF YOU ALREADY HAVE A FILE OF "
733 PRINT"RAT BEHAVIOR ON THIS DISK, IT WILL BE "
734 PRINT"REPLACED BY THE CURRENT FILE. ENTER Y IF SUCH A FILE ALREADY
    EXISTS."
735 INPUT NZ$:IF NZ$="Y" THEN OPEN 15,8,15:PRINT#15,"S0:BEHAVIOR":
    CLOSE15
740 OPEN 2,8,2,"BEHAVIOR,S,W":FOR I=1 TO 79
750 PRINT#2,F(I):NEXT:CLOSE 2
760 PRINT:INPUT"WANT TO STOP (Y/N)";BD$
770 IF LEFT$(BD$+"Y",1)<>"Y" THEN 135
790 :::::::::::::::::::::::::::::::::::::
```

This program represents an attempt to make a computer simulate behavior, the behavior of a rat in a Skinner box. Here is a line-by-line explanation of how it works.

LINE	EXPLANATION
10-60	These lines identify the program and provide credit information.

70-160	This is the main program. Line 70 is a remark statement.
80	This line dimensions various arrays. Y is the array that stores behavior codes. P is the starting point in the array for a given behavior, and S is the number of possible behaviors that follow P for each behavior. F is the frequency of the behavior, and A$ is the actual behavior.
90	This line reads in the elements of array Y and sets each element of array F to 1.
100	This line reads in the P and S arrays.
110	This line reads in the behaviors from DATA statements 480-670 into the A$ array.
120	This line calls a subroutine to print the instructions.
130	This sets the first exhibited behavior as 3, which is RAT STANDS ON ALL FOUR LEGS.
135	K is the random choice of P(X) behaviors starting at S(X).
136-137	These lines add up the frequencies of possible subsequent behaviors.
140	This line selects a behavior based on its frequency. If the behavior selected is within the frequency range associated with it, it is printed and control goes to line 150.
145	This line return control to line 135 where another possible behavior is selected.
150	This line sets a timer at zero and sets R$ to the empty string.
152	This line reads the keyboard for a key pressed within 5 seconds.
153	If the pressed key is f1, control goes to line 700.
155	If any key was pressed, the frequency of the behavior is increased by 10%. X becomes the number of the new behavior, and control goes to line 135.
160	No key was pressed; the frequency of the present behavior is decreased, X is assigned the new behavior number, and control goes to line 135.
199	This is a separation line.
200-300	This subroutine prints the instructions and asks if you want to load data from cassette tape.
310	A separation line.
400-470	These DATA statements contain the 79 numbers for the Y array. The next 40 numbers are the P and S numbers, as P(1),S(1),P(2),S(2), etc.
480-670	Each of these lines represents an element of the A$ array. For example, A$(1) is RAT APPROACHES PELLET DISPENSER.



```
                         REINFORCEMENT


     THIS PROGRAM SIMULATES A SKINNER
BOX. THE DESCRIPTION OF THE BEHAVIOR OF
THE RAT WILL BE DISPLAYED ON THE
SCREEN. BY PRESSING ANY KEY, YOU CAN
POSITIVELY REINFORCE THAT BEHAVIOR.
POSITIVELY REINFORCED BEHAVIOR WILL
INCREASE IN FREQUENCY.

     PRESS F1 TO STOP OR SAVE BEHAVIOR
ON CASSETTE TAPE.

DO YOU WANT TO ENTER BEHAVIOR <Y/N>? N
PRESS THE RETURN KEY TO CONTINUE. OK
     THE RAT HAS JUST BEEN PLACED IN
THE SKINNER BOX.


RAT WALKS AROUND ON RIGHT SIDE OF BOX.

RAT WALKS AROUND ON RIGHT SIDE OF BOX.

THE RAT LIES DOWN.

IT CONTINUES LYING DOWN.

RAT CONTINUES SLEEPING.

THE RAT WAKES UP.

IT GOES TO SLEEP.

RAT CONTINUES SLEEPING.

RAT CONTINUES SLEEPING.

RAT CONTINUES SLEEPING.

THE RAT WAKES UP.

IT GOES TO SLEEP.

RAT CONTINUES SLEEPING.

THE RAT WAKES UP.

RAT STANDS ON ALL FOUR LEGS.

RAT CONTINUES STANDING ON ALL FOURS.

RAT WALKS AROUND ON RIGHT SIDE OF BOX.

THE RAT FACES THE BAR PRESS ON HIND LEGS.
```

Fig. 3-1. A display of simulated computer behavior from the Training Rat Program.

680	A separation line.
700-770	This subroutine asks you if you want to save the current behavior on tape for use later. It then asks you if you want to stop the program. The computer responds accordingly.
790	The last line of the program.

Program Operation. Figure 3-1 is a display of the program as it runs on the Commodore 64. After the initial instructions and questions, the program starts displaying the behaviors. Your job is to press any key at an appropriate time to reinforce the behavior. In this way you increase the frequency that it will occur. At the beginning, all the behaviors associated with the particular behavior have an equal chance of occurring.

You can train the rat to go through a series of behaviors by reinforcing the proper behaviors.

BOOTSTRAP SYSTEMS

We now look at an interesting sort of ideas. What would happen if you could write a computer program that would change its programming as it ran? This is possible on the Commodore 64 because we can find where and how the programs are stored. By writing a careful program, we can have the computer change its programming as it runs.

Let's define what a bootstrap system is. A bootstrap system is any computer program that creates new program statements as it runs. This is not to be confused with bootable files, that load the operating system or language for some computers. The bootstrap system is any program that changes itself as it runs. We will look at two simple examples and how they work.

On the Commodore 64 the BASIC programs that you enter into RAM are stored as *tokens.* Tokens are an internal form that is neither the letters and numbers of BASIC nor executable machine language code. This intermediate form is a series of numbers from 0 to 255 stored in the RAM and represents the program. It occupies less space than the entire collection of characters comprising the BASIC language commands and operators. Figure 3-2 is a partial list of the codes associated with that internal form. Thus BASIC commands are stored as 8-bit numbers in memory.

The First Bootstrap Program

We now look at a bootstrap program that will define the variable A as 1 and will print it every other time that the program is run. Listing 3-3 includes three versions of the program. The first version shows how the program should be entered into the Commodore 64. The second version shows how the program appears

128	END	166	SOC<
129	FOR	167	THEN
130	NEXT	168	NOT
131	DATA	169	STEP
132	INPUT+	170	+
133	INPUT	171	−
134	DIM	172	*
135	READ	173	/
136	LET	174	↑
137	GOTO	175	AND
138	RUN	176	OR
139	IF	177	>
140	RESTORE	178	=
141	GOSUB	179	<
142	RETURN	180	SGN
143	REM	181	INT
144	STOP	182	ABS
145	ON	183	USR
146	WAIT	184	FRE
147	LOAD	185	POS
148	SAVE	186	SQR
149	VERIFY	187	END
150	DEF	188	LOG
151	POKE	189	EXP
152	PRINT+	190	COS
153	PRINT	191	SIN
154	CONT	192	TAN
155	LIST	193	ATN
156	CLR	194	PEEK
157	CMD	195	LEN
158	SYS	196	STR$
159	OPEN	197	VAL
160	CLOSE	198	ASC
161	GET	199	CHR$
162	NEW	200	LEFT$
163	TAB<	201	RIGHT$
164	TO	202	MID$
165	FN	203	GO

Fig. 3-2. Code numbers and tokens for BASIC commands as stored in the Commodore 64.

Listing 3-3 The First Bootstrap Program

```
0 ::::::::::::::::::::::
10 REM  BOOTSTRAP PROGRAM - PROGRAM 8
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
50 POKE2053,65:POKE2054,178:POKE2055,49
60 POKE2057,153:POKE2058,65
70 POKE2060,151:POKE2061,50:POKE2062,48
```

```
80 POKE2063,53:POKE2064,51:POKE2065,44
90 POKE2066,49:POKE2067,52:POKE2068,51
100 POKE2070,128
```

```
0 A=1:PRINTA:POKE2053,143:END::
10 REM  BOOTSTRAP PROGRAM - PROGRAM 8
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
50 POKE2053,65:POKE2054,178:POKE2055,49
60 POKE2057,153:POKE2058,65
70 POKE2060,151:POKE2061,50:POKE2062,48
80 POKE2063,53:POKE2064,51:POKE2065,44
90 POKE2066,49:POKE2067,52:POKE2068,51
100 POKE2070,128
```

```
0 REM=1:PRINTA:POKE2053,143:END::
10 REM  BOOTSTRAP PROGRAM - PROGRAM 8
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
50 POKE2053,65:POKE2054,178:POKE2055,49
60 POKE2057,153:POKE2058,65
70 POKE2060,151:POKE2061,50:POKE2062,48
80 POKE2063,53:POKE2064,51:POKE2065,44
90 POKE2066,49:POKE2067,52:POKE2068,51
100 POKE2070,128
```

after it is run once, and the third version is a listing of the program after it has been run a second time. Subsequent runs of the program alternate between the second and the third version.

This program is unique in that the program operates in two different ways. When the program is first run, it creates a new program statement from line 0. This particular program is simple. It creates a subprogram out of itself. That subprogram defines the variable A and 1 and prints out that value. It alternates between two different modes of operation.

This program by itself is not spectacular, but it should lead you to explore ways of having programs change themselves. Remember the cellular automaton program that changes the contents of the DATA statements as it ran? This concept, the idea of bootstrap systems, has some intriguing possibilities.

Here's a program line-by-line explanation of the program.

LINE	**EXPLANATION**
0	This line consists of 20 colons and will be used to store a new program line.
10-40	These lines are the title and credit data for the program.
50	POKE 2053,65 changes the first colon to the letter A.

	POKE 2054,178 changes the second colon to the = character. POKE 2055,49 changes the third colon to the number 1.
60	This line changes the fifth colon to PRINT and the sixth colon to the letter A.
70	This line changes the eighth, ninth, and tenth colons to POKE, 2, and 0, respectively.
80	This line changes the next three colons to 5, 3, and the comma character.
90	These three statements change the next three colons to the digits 1, 4, and 3.
100	This POKE command changes the 18th colon to END.

Program Operation. As the program is run, the contents of line 0 change. The first time it runs, some of the colons of line 0 are changed to BASIC statements. Actually what happens is that we have altered the contents of the RAM area that stores the tokens of the program. The next time the program is run, only line 0 is executed because one of the tokens added was the END command, code 128. When this line is run, the token at RAM position 2053, the A letter, is changed to REM, the remark statement. That means the next time that the program is run, line 0 will not be executed. The other program lines will be executed, as in the first version. The program alternates between the last two forms from then on.

The Second Bootstrap Program

Listing 3-4 shows the two forms of another program that changes the line statements. This program will run only twice. The first time it is run, it creates the statements of line 0, shown in the second version. The second time it is run, it executes line 0, which also erases the program. This program will run, run again, and then disappear.

Listing 3-4 The Second Bootstrap Program

```
0 ::::::::::::::::::::
10 REM BOOTSTRAP PROGRAM - PROGRAM 9
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
50 IFA=0THENPOKE2053,65:POKE2054,178:POKE2055,49
60 IFA=0THENPOKE2057,153:POKE2058,65
70 IFA=0THENPOKE2060,151:POKE2061,50:POKE2062,48
80 IFA=0THENPOKE2063,53:POKE2064,48:POKE2065,44
90 IFA=0THENPOKE2066,48

0 A=1:PRINTA:POKE2050,0::::::
10 REM BOOTSTRAP PROGRAM - PROGRAM 9
```

```
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
50 IFA=0THENPOKE2053,65:POKE2054,178:POKE2055,49
60 IFA=0THENPOKE2057,153:POKE2058,65
70 IFA=0THENPOKE2060,151:POKE2061,50:POKE2062,48
80 IFA=0THENPOKE2063,53:POKE2064,48:POKE2065,44
90 IFA=0THENPOKE2066,48
```

Here is a brief explanation of the lines in the first version.

LINE	**EXPLANATION**
0	This line contains 20 colons, which provide room for storing the new statements.
10-40	The title and credits of the program.
50-90	These lines run only if A is 0, in other words, if it is the first time that the program is run. These lines generate the BASIC statements in line 0 of the second form. Line 0 in that program assign A as 1, print it and set the first A in that line equal to a code of 0, a null code, which erases the lines of the program.

Program Operation. After entering the program into memory, type LIST. You should see the first form of the program. After you run the program, type LIST. You should see the second form of the program. If you type RUN again, the computer will print 1 on the screen and erase the program. When you type LIST, the computer will display nothing but the READY prompt. You MIGHT recover the program by typing POKE 2053,65. If the RAM is disturbed, it generally won't reinstate the program in its entirety.

Imagine how this concept could be expanded. Certain lines of a program could be activated to dimension arrays, erase lines at the end of the program, destroy line statements forever by converting them to colons, and inactivate lines by making the first token REM. A good program could generate all sorts of fascinating commands. Experiment!

Chapter 4

Natural Language Processing

One of the most active areas of artificial intelligence is the area of natural language processing. In natural language processing, the computer reacts to English statements. The ideal natural language processor would interpret commands as a person would. Programming would be as easy as teaching a person, maybe easier. The point of natural language is to have the computer understand you, instead of you trying to understand the computer. Obviously English is only one of several languages that could be used in programming computers to understand natural language.

In this chapter we will explore two programs that utilize natural language processing. The first is a program based on *ELIZA,* a natural language program originally written by Joseph Weizenbaum at the Massachusetts Institute of Technology in 1966. The second BASIC program is called PERFECT LOGIC, a 1984 original by me that makes conclusions based on set theory. (I consider this program to be the best in the book.)

DEFINITIONS AND EXAMPLES

Let's talk about natural language and give some examples. When we solve problems, we must find a way from the question to the answer. We might think that the answer to a question or the solution of a problem is somehow a function of that question or problem. Our task in finding the answer is to break down the question into a series of steps. If we know the solution to a few of the steps, we are that much closer to the answer to the overall question. This same approach can be used in artificial intelligence. What the computer might be programmed to do is to save some of the solutions to steps in the problems that it encounters. This stored knowledge would be used to break new problems down into similar steps and to organize memory properly to solve the problem.

In the case of natural language, words are the steps of the problem. The arrangement of the words and statements determine how the computer is to respond. If the computer "remembers" what action it is to take when it encounters a sequence of words, the problem is on its way to being solved. This will all become clearer by the end of this chapter.

English-Like Conversations

The Commodore 64 can be programmed to respond correctly to a limited set of English sentences and questions. If you have a program that allows the computer to store some of the statements or questions that you entered earlier, you might be able to have somewhat of a meaningful conversation. Unfortunately, the computer's world is very limited. The only way the computer "knows" how to respond is by the contents of its memory. It is aware of nothing and only responds electronically to the signals that it receives. To the human observer, however, well written programs can seem incredible.

In the first program in this chapter the computer will look for key words and then attempt to respond, based on the content of those key words. Key words would be the link between the question and the answer. The second program involves the creation of sets and subsets, and the use of an English language interpreter to place nouns in sets. You can think of graph theory when you look at these examples. Trees can be used to trace a path from one point to another to see if there is a connection, which would indicate a solution.

An example of a program that used natural language to communicate to a computer is *SHRDLU,* written by Terry Winograd at MIT. In that program the computer manipulated a collection of blocks, pyramids, and boxes. This 1971 program would attempt to respond to English commands and perform certain actions on the objects within its mathematical domain. If the computer was uncertain about the request, it would question the user for information that would clarify the request. This program would then be able to process natural language, manipulate actions, and query the user when a request was ambiguous. The result was remarkable.

One of the harder things to do on the computer is to program it to understand "common sense" because the computer's world is very limited. At this point in time, personal computers have neither the memory to store enough data nor the advanced programming capabilities that would allow them to deal with things outside of their limited world. The computer is at best an expert in its own little world, and common sense is foreign to them.

The ability to communicate with computers using natural language becomes quite a challenge for the programmer. The approach that the programmer uses is determined by what he wants the computer to accomplish.

Compositional Works

One idea that we will not explore but that is worth discussing is computer composition. Here the computer strings together nouns, verbs, phrases, and sentences. Often the programs produce ridiculous sentences. However, if the verbs can be made to match the nouns and phrases, they can be somewhat meaningful. Generally programs of this type use the random number generator to select arbitrary words and phrases.

Do you remember how one state of our cellular automaton program determined the next state? If a program could be devised to construct compositional works based on words and phrases, we might have an interesting text generator. One sentence would lead to another to form paragraphs, and then chapters. This concept might not be as farfetched as it sounds. Remember that in the cellular automaton you could not predict precisely how the figure would appear, even though it used only a few simple rules and a starting pattern. A compositional automaton could work the same way.

THE STORY OF ELIZA

Because the first program in this chapter is like *ELIZA*, it may be worthwhile to discuss *ELIZA*. At one time *ELIZA* was the most famous computer program in the world. It has now been supplanted by VisiCalc, the electronic spreadsheet marketed by VisiCorp. Nevertheless, ELIZA was innovative in allowing the first uses of natural language in a real time environment.

ELIZA was intended as a simulation of a nondirective psychotherapy session based on the techniques of Carl Rogers. The computer program would search through the sentences or questions that the user would enter and from those entries select key words or phrases from which to print a reply. By the clever use of grammar and the rearrangement of sentence structure and so forth, the computer would be able to produce a somewhat meaningful response. The computer program required a long list of key words and a preprogrammed set of instructions on how to react. For instance, if the user typed the word sister in a sentence, the computer would ask the user to tell him some more about his family. If no key words were found, the program would default to printing from a selection of stock, noncommittal replies.

ELIZA was really just an experiment in how to make people think that a computer could think and understand what they were talking about. Many times the program would produce nonsense, especially when the program stored a sentence incorrectly. When conversing with the computer, people would use incomplete sentences—garbage in, garbage out. The program was interesting, nevertheless.

A program like *ELIZA* was apt to get out of hand. One of the unfortunate effects of *ELIZA* was that people took the program seriously. The original program was never intended as anything other than an experiment. It was never intended to replace a psychotherapist. People tended to regard the computer as a personality and shared their personal feelings with the computer. They were caught by the illusion. Beware of this attitude as you examine any artificial intelligence programs.

LANGUAGE PROCESSING IN BASIC

The next two programs are written in BASIC and run well on the Commodore 64. I will explain them both in detail.

An ELIZA-Like Program

The program in Listing 4-1 is like the ELIZA program discussed previously. This one is written specifically for the Commodore 64 and contains features used only on the Commodore 64. The computer's responses are written in white, and the user's responses are in cyan. The computer reads the keyboard for input (including final punctuation marks) and does not display question marks as a prompt.

Listing 4-1 The *ELIZA*-like Program

```
10 ::::::::::::::::::::::::::::::::::::
20 REM    AN "ELIZA"-LIKE PROGRAM
30 REM WRITTEN BY TIMOTHY J. O'MALLEY
40 REM COPYRIGHT 1984, TAB BOOKS INC.
50 REM (WRITTEN FOR THE COMMODORE 64)
60 ::::::::::::::::::::::::::::::::::::
70 REM       *** MAIN PROGRAM ***
80 GOSUB 170:REM SET UP PROGRAM
90 GOSUB 250:REM INPUT SUBROUTINE
100 GOSUB 390:REM SYNONYMS AND ANTONYMS
110 GOSUB 570:REM LOOK FOR KEYWORDS
120 GOSUB 1120:REM REMOVE MARKERS
130 GOSUB 1190:REM COMPUTER'S REPLY
```

```
140 GOTO 90:REM ENDLESS LOOP
150 ::::::::::::::::::::::::::::::::::::::
160 REM     *** SET UP PROGRAM ***
170 M=50:REM NUMBER OF SYNONYMS & ANTONYMS
180 U=84:REM NUMBER OF POSSIBLE REPLIES
190 DIM MY$(200):MC=0:REM STACK OF "MY" STATEMENTS
200 PRINT "{CLR}";:PRINT "{WHT}";:REM CLEAR SCREEN AND SET COMPUTER'S
    COLOR AS WHITE
210 PRINT "      HELLO. WHAT'S ON YOUR MIND?":PRINT
220 RETURN
230 ::::::::::::::::::::::::::::::::::::::
240 REM   *** INPUT SUBROUTINE ***
250 RESTORE:REM ALLOW DATA TO BE RE-READ
260 X=0:A$="":PRINT "{CYAN}";:REM KEYWORD FLAG=0 & YOUR INPUT COLOR IS
    CYAN
270 OPEN 50,0:REM OPEN THE KEYBOARD FILE TO "READ" KEYBOARD
280 GET#50,Z$:IF Z$="" THEN 280:REM RE-READ KEYBOARD
290 IF Z$=CHR$(13) THEN PRINT:CLOSE50:PRINT "{WHT}";:GOTO 320:REM END
    INPUT
300 IF Z$=CHR$(20) THEN PRINT Z$;:A$=LEFT$(A$,LEN(A$)-1):GOTO 280:REM
    DEL CHAR
310 A$=A$+Z$:PRINT Z$;:GOTO 280:REM CONCATENATE INPUT STRING & DISPLAY
    CHAR
320 IF A$="" THEN PRINT "WHAT'S THE PROBLEM?":X=1:RETURN:REM NO ENTRY
330 IF A$=R$ THEN PRINT "YOU ARE BEING REPETITIOUS.":X=1:RETURN
340 R$=A$:REM STORE LAST INPUT
350 A$=" "+LEFT$(A$,LEN(A$)-1)+" ":REM ERASE PUNCTUATION AND ADD
    SPACES
360 RETURN
370 ::::::::::::::::::::::::::::::::::::::
380 REM *** SYNONYMS AND ANTONYMS ***
390 IF X THEN RETURN:REM NO VALID INPUT
400 FOR I=1 TO M/2
410 READ E$,N$:REM READ WORD PAIR
420 FOR S=1 TO LEN(A$)-LEN(E$)+1
430 IF MID$(A$,S,1)<>" " THEN 530:REM MIDDLE OF WORD
440 IF E$=MID$(A$,S,LEN(E$)) THEN A$=LEFT$(A$,S-1)+N$+MID$(A$,S+LEN
    (E$))
450 DATA " MOM "," MOTHER "," DAD "," FATHER "," KIDS "," CHILDREN "
460 DATA " DREAMS "," DREAM "," KID "," CHILD "," HOUSE "," HOME* "
470 DATA " I "," YOU@ "," YOU "," I "," ME "," YOU "," ONE "," 1 "
480 DATA " MY "," YOUR* "," TWO "," 2 "," THREE "," 3 "
490 DATA " YOUR "," MY "," MYSELF "," YOURSELF* "," TOO "," ALSO "
500 DATA " YOURSELF "," MYSELF "," HURT "," HARM "," HOME "," HOUSE* "
510 DATA " I'M "," YOU'RE* "," YOU'RE "," I'M "," AM "," ARE@ "
520 DATA " WERE "," WAS "," EASY "," SIMPLE "," DIFFICULT "," HARD "
530 NEXT S,I
540 RETURN
550 ::::::::::::::::::::::::::::::::::::::
560 REM   *** LOOK FOR KEYWORDS ***
570 IF X THEN RETURN
580 FOR I=1 TO U
590 READ E$,J
```

```
600 FOR S=1 TO LEN(A$)-LEN(E$)+1
620 IF E$=MID$(A$,S,LEN(E$)) THEN R$=MID$(A$,S+LEN(E$)):GOTO 1910
630 NEXT S,I
640 GOTO 950:REM NO KEYWORDS LOCATED
650 R$=LEFT$(R$,LEN(R$)-1):RETURN
660 DATA "COMPUTER",1,"MACHINE",1
670 DATA " NAME ",2,"ALIKE",3," LIKE ",3," SAME ",3
680 DATA "YOU@ REMEMBER",4,"DO I REMEMBER",5,"YOU@ DREAMED",6
690 DATA " DREAM ",7," IF ",8,"EVERYBODY",9,"EVERYONE",9
700 DATA "NOBODY",9,"NO ONE",9,"WAS YOU@",10,"YOU@ WAS",11
710 DATA "WAS I",12,"YOUR* MOTHER",13,"YOUR* FATHER",13
720 DATA "YOUR* HUSBAND",13,"YOUR* CHILDREN",13,"YOUR*",14
730 DATA "YOUR* SISTER",13,"YOUR* BROTHER",13,"YOUR* WIFE",13
740 DATA "ALWAYS",15,"ARE I",16,"ARE@ YOU@",18," HOW ",25
750 DATA "BECAUSE",19,"CAN I",20,"CAN YOU@",21,"CERTAINLY",22
760 DATA "DEUTSCH",23,"ESPANOL",23,FRANCAIS",23,"HELLO",24
770 DATA "I REMIND YOU OF",3,"I ARE",26,"I'M",26
780 DATA "ITALIANO",23,"MAYBE",28," MY ",29," NO ",30
790 DATA "PERHAPS",28,"SORRY",31,"WHAT ",25,"WHEN ",25
800 DATA "WHY DON'T I",32,"WHY CAN'T YOU@",33,"YES",22
810 DATA "YOU@ WANT",34,"YOU@ NEED",34," ARE ",17," I ",27
820 DATA "YOU@ ARE@ SAD",35,"YOU'RE* SAD",35
830 DATA "YOU@ ARE@ UNHAPPY",35,"YOU'RE* UNHAPPY",35
840 DATA "YOU@ ARE@ DEPRESSED",35,"YOU'RE* DEPRESSED",35
850 DATA "YOU@ ARE@ SICK",35,"YOU'RE* SICK",35
860 DATA "YOU@ ARE@ HAPPY",36,"YOU'RE* HAPPY",36
870 DATA "YOU@ ARE@ ELATED",36,"YOU'RE* ELATED",36
880 DATA "YOU@ ARE@ GLAD",36,"YOU'RE* GLAD",36
890 DATA "YOU@ ARE@ BETTER",36,"YOU'RE* BETTER",36
900 DATA "YOU@ FEEL YOU@",37,"YOU@ THINK YOU@",37
910 DATA "YOU@ BELIEVE YOU@",37,"YOU@ WISH YOU@",37
920 DATA " YOU@ ARE@",38,"YOU'RE*",38,"YOU@ CAN'T",39
930 DATA "YOU@ CANNOT",39,"YOU@ DON'T",40,"YOU@ FEEL",41
940 DATA " HE ",42," SHE ",43
945 ::::::::::::::::::::::::::::::::::::
950 REM       *** NO KEYWORDS ***
960 X=1:IF MC=0 THEN 980
970 RAN=1+INT(5*RND(1)):ON RAN GOTO 980,1030,1030,1030,980
980 RAN=1+INT(4*RND(1)):ON RAN GOTO 990,1000,1010,1020
990 PRINT"I AM NOT SURE I UNDERSTAND YOU FULLY.":RETURN
1000 PRINT"PLEASE GO ON.":RETURN
1010 PRINT"WHAT DOES THAT SUGGEST TO YOU?":RETURN
1020 PRINT"DO YOU FEEL STRONGLY ABOUT DISCUSSING SUCH THINGS?":RETURN
1030 Y$=MY$(1+INT(MC*RND(1))):RAN=1+INT(6*RND(1))
1040 ON RAN GOTO 1050,1060,1070,1080,1090,1100
1050 PRINT"LET'S DISCUSS FURTHER WHY YOUR"+Y$+".":RETURN
1060 PRINT"EARLIER YOU SAID YOUR"+Y$+".":RETURN
1070 PRINT"DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR"+Y$+"
    .":RETURN
1080 PRINT"TELL ME MORE ABOUT WHY YOUR"+Y$+".":RETURN
1090 PRINT"IS IT REALLY TRUE THAT YOUR"+Y$+"?":RETURN
1100 PRINT"LET'S SEE. YOU TOLD ME YOUR"+Y$+". CARE TO ELABORATE?":
    RETURN
1105 ::::::::::::::::::::::::::::::::::::
```

```
1110 REM *** REMOVE @ AND * MARKERS ***
1120 IF X=1 THEN RETURN
1130 FOR S=1 TO LEN(R$)
1140 G$=MID$(R$,S,1):IF G$="@" OR G$="*" THEN R$=LEFT$(R$,S-1)+MID$(R$
     ,S+1)
1150 NEXT S
1160 RETURN
1170 :::::::::::::::::::::::::::::::::::
1180 REM   *** COMPUTER'S REPLY ***
1190 IF X=1 THEN RETURN
1200 IF J<11 THEN ON J GOTO 1310,1320,1330,1340,1350,1360,1370,1380,
     1390,1400
1210 J=J-10
1220 IF J<11 THEN ON J GOTO 1410,1420,1430,1440,1450,1460,1470,1480,
     1490,1500
1230 J=J-10
1240 IF J<11 THEN ON J GOTO 1510,1520,1530,1540,1550,1560,1570,1580,
     1590,1600
1250 J=J-10
1260 IF J<11 THEN ON J GOTO 1610,1660,1670,1680,1690,1700,1710,1720,
     1730,1740
1270 J=J-10
1280 ON J GOTO 1750,2070,2080
1290 :::::::::::::::::::::::::::::::::::
1300 REM *** COMPUTER'S REPLIES ***
1310 PRINT"DO COMPUTERS WORRY YOU?":RETURN
1320 PRINT"I AM NOT INTERESTED IN NAMES.":RETURN
1330 PRINT"IN WHAT WAY?":RETURN
1340 PRINT"DO YOU OFTEN THINK OF"R$"?":RETURN
1350 PRINT"DID YOU REALLY THINK I WOULD FORGET"R$"?":RETURN
1360 PRINT"REALLY, "R$"?":RETURN
1370 PRINT"WHAT DOES THAT DREAM SUGGEST TO YOU?":RETURN
1380 PRINT"DO YOU THINK THAT IT'S LIKELY THAT IF"R$"?":RETURN
1390 PRINT"REALLY, "E$"?":RETURN
1400 PRINT"WHAT IF YOU WERE"R$"?":RETURN
1410 PRINT"WERE YOU REALLY?":RETURN
1420 PRINT"WOULD YOU LIKE TO BELIEVE I WAS"R$"?":RETURN
1430 PRINT"TELL ME MORE ABOUT YOUR FAMILY.":RETURN
1440 GOSUB 1960:GOSUB 1780:RETURN
1450 PRINT"CAN YOU THINK OF A SPECIFIC EXAMPLE?":RETURN
1460 PRINT"WHY ARE YOU INTERESTED IN WHETHER I AM"R$" OR NOT?":RETURN
1470 PRINT"DID YOU THINK THEY MIGHT NOT BE "R$"?":RETURN
1480 PRINT"DO YOU BELIEVE YOU ARE"R$"?":RETURN
1490 PRINT"IS THAT THE REAL REASON?":RETURN
1500 PRINT"YOU BELIEVE I CAN"R$", DON'T YOU?":RETURN
1510 PRINT"WHETHER YOU CAN"R$" DEPENDS ON YOU MORE THAN ON ME.":RETURN
1520 PRINT"YOU SEEM QUITE SURE.":RETURN
1530 PRINT"SORRY, I SPEAK ONLY ENGLISH.":RETURN
1540 PRINT"HOW DO YOU DO?":RETURN
1550 PRINT"WHY DO YOU ASK?":RETURN
1560 PRINT"WHAT MAKE YOU THINK I AM"R$"?":RETURN
1570 PRINT"LET'S NOT TALK ABOUT ME.":RETURN
1580 PRINT"YOU ARE CERTAIN?":RETURN
```

```
1590 PRINT"WHY ARE YOU CONCERNED OVER MY "R$"?":RETURN
1600 PRINT"ARE YOU SAYING 'NO' JUST TO BE NEGATIVE?":RETURN
1610 RAN=1+INT(4*RND(1)):ON RAN GOTO 1620,1630,1640,1650
1620 PRINT"PLEASE DON'T APOLOGIZE.":RETURN
1630 PRINT"APOLOGIES ARE NOT NECESSARY.":RETURN
1640 PRINT"WHAT FEELING DO YOU HAVE WHEN YOU APOLOGIZE?":RETURN
1650 PRINT"YOU NEEDN'T FEEL YOU HAVE TO APOLOGIZE.":RETURN
1660 PRINT"DO YOU BELIEVE I DON'T"R$"?":RETURN
1670 PRINT"DO YOU THINK YOU SHOULD BE ABLE TO"R$"?":RETURN
1680 PRINT"WHAT WOULD IT MEAN TO YOU IF YOU GOT"R$"?":RETURN
1690 GOSUB 1840:GOSUB 1880:PRINT"I'M SORRY TO HEAR YOU ARE"R$".":
     RETURN
1700 GOSUB 1840:GOSUB 1880:PRINT"HOW HAVE I HELPED YOU BE"R$"?":RETURN
1710 PRINT"DO YOU REALLY THINK SO?":RETURN
1720 PRINT"IS IT BECAUSE YOU ARE"R$" THAT YOU CAME TO ME?":RETURN
1730 PRINT"HOW DO YOU KNOW YOU CAN'T"R$"?":RETURN
1740 PRINT"DON'T YOU REALLY"R$"?":RETURN
1750 PRINT"TELL ME MORE ABOUT SUCH FEELINGS.":RETURN
1760 ::::::::::::::::::::::::::::::::::::
1770 REM    *** KEYWORD IS "MY" ***
1780 IF LEN(R$)<12 THEN RETURN
1790 MC=MC+1:ID=MC:IF MC>200 THEN MC=200:ID=1+INT(200*RND(1))
1800 MY$(ID)=R$
1810 RETURN
1820 ::::::::::::::::::::::::::::::::::::
1830 REM    *** REMOVE @ MARKER ***
1840 IF MID$(E$,4,1)="@" THEN R$=RIGHT$(E$,LEN(E$)-9)
1850 RETURN
1860 ::::::::::::::::::::::::::::::::::::
1870 REM    *** REMOVE * MARKER ***
1880 IF MID$(E$,7,1)="*" THEN R$=RIGHT$(E$,LEN(E$)-7)
1890 RETURN
1900 ::::::::::::::::::::::::::::::::::::
1910 IF R$<>"" THEN 650
1920 RETURN
1950 ::::::::::::::::::::::::::::::::::::
1960 RAN=1+INT(8*RND(1))
1970 ON RAN GOTO 1980,1990,2000,2010,2020,2030,2040
1980 PRINT"YOUR"R$"?":GOSUB 1780:RETURN
1990 PRINT"WHY IS IT THAT YOUR"R$"?":GOSUB 1780:RETURN
2000 PRINT"YOUR"R$"!":GOSUB 1780:RETURN
2010 PRINT"HOW IS IT THAT YOUR"R$"?":GOSUB 1780:RETURN
2020 PRINT"YOUR"R$"? REALLY?":GOSUB 1780:RETURN
2030 PRINT"MY"R$", TOO.":GOSUB 1780:RETURN
2040 PRINT"WHOSE"R$"?":GOSUB 1780:RETURN
2050 PRINT"YOU SAY YOUR"R$"?":GOSUB 1780:RETURN
2060 ::::::::::::::::::::::::::::::::::::
2070 PRINT"DOES HE HAVE A PROBLEM?":RETURN
2080 PRINT"TELL ME MORE ABOUT HER.":RETURN
2090 ::::::::::::::::::::::::::::::::::::
    HELLO. WHAT'S ON YOUR MIND?

I HAVE THIS PROBLEM.

DO YOU FEEL STRONGLY ABOUT DISCUSSING SUCH THINGS?

HUH? MY DOG HAS FLEAS.

HOW IS IT THAT YOUR DOG HAS FLEAS?

I THINK MY DOG IS OUTSIDE TOO MUCH.

WHOSE DOG IS OUTSIDE ALSO MUCH?

MY STUPID DOG IS OUTSIDE TOO OFTEN.

WHOSE STUPID DOG IS OUTSIDE ALSO OFTEN?

MINE IS.

PLEASE GO ON.

HE IS MISERABLE.

DOES HE HAVE A PROBLEM?

YOU WEREN'T LISTENING TO ME.

LET'S NOT TALK ABOUT ME.

MY COURSE IN QUANTUM MECHANICS IS DIFFICULT.

MY COURSE IN QUANTUM MECHANICS IS HARD, TOO.

DO YOU TAKE COURSES?

LET'S NOT TALK ABOUT ME.

MY HOUSE IS TOO NOISY.

WHOSE HOME IS ALSO NOISY?

WHAT TIME IS IT NOW?

WHY DO YOU ASK?

I HATE MACHINES LIKE YOU.

DO COMPUTERS WORRY YOU?

NO.

ARE YOU SAYING 'NO' JUST TO BE NEGATIVE?

BUZZ OFF!

WHAT DOES THAT SUGGEST TO YOU?
```

Fig. 4-1. The start of a conversation with an "ELIZA"-like program.

```
BUGS.

EARLIER YOU SAID YOUR DOG HAS FLEAS.

SOME PEOPLE BUG ME.

IS IT REALLY TRUE THAT YOUR STUPID DOG IS OUTSIDE ALSO OFTEN?

YES.

YOU SEEM QUITE SURE.
```

Fig. 4-1. *(continued)*

As shown in the sample program run in Fig. 4-1 this program provides an English conversation with the user and stores some of the entries of the user, namely those containing the word MY. This program should provide an interesting amusement for the user. In the explanation of the program lines below I have given some suggestions on how to change the program. You might also want to save some of the stored MY$ responses on tape of disk for later use.

Here is a detailed description of the lines of the program.

LINE	**EXPLANATION**
10-60	This is the identification information for the program.
70-140	This is the main program. Line 70 is a REM statement identifying this section of the program.
80	This line calls the subroutine in lines 160-220.
90	The subroutine called in this line allows you to enter responses and changes the color of the text depending on who is responding, you or the computer.
100	This line calls a subroutine that looks for synonyms and antonyms of words used by the user.
110	This line calls a subroutine that searches for key words or key phrases.
120	This line calls a subroutine that removes markers the program placed on the antonyms. They are placed to keep the computer from changing altered words back to the original words.
130	This line calls a subroutine that prints out the computer's replies.
140	This statement establishes an endless loop that cycles back to line 90, where new input is requested from the user.

150	This line acts as a separation between parts of the program.
160–220	This subroutine sets up the program and is run only at the beginning of the program.
160	This remark tells what this section does.
170	There are 25 synonym and antonym pairs. M is the variable that stores that value.
180	U is a variable set at 84 and used as the number of possible replies.
190	This line is unique. The MY$ array, or "stack," stores statements input by the user that contain the word MY. This line dimensions that string array and sets the stack pointer, MC, equal to zero.
200	This line prints the CLR character and the WHT character. White is the color that the computer will use when printing lines on the screen.
210	This is the computer's opening question.
220	The program control returns to the main program from the subroutine.
230	This line separates the sections of the program.
240–360	This subroutine allows for input from the user.
240	This line identifies this subroutine as the input subroutine.
250	RESTORE allows the data in the DATA statements to be used over and over.
260	This line sets a *flag,* X, to 0, sets A$ to an empty string, and changes the color to CYAN, which is the user's text color.
270	This statement opens the "file" corresponding to the keyboard, allowing the computer to scan letters that the user is typing without resorting to INPUT statements.
280	This line actually scans the keyboard for letters. One letter at a time is stored in Z$.
290	If the user presses the RETURN key (character code *13), the keyboard file is closed and the color changes* back to white. The program branches to line 320.
300	If the key pressed is the DEL character, the computer erases the last entry. Because the sentences or questions are made into the string, A$, the last character entered into that is deleted also. The program then branches to line 280 for more characters.
310	A$ equals A$ *concatenated with the last letter typed. The last letter typed is printed on the screen. The* program then goes to line 280 for more letters.
320	If the user simply presses the RETURN key, he is

	asked what the problem is. The program then returns to the main program.
330	If the user enters the same reply twice, the computer states that he is being repetitious. The program then returns to the main program.
340	R$ is set as A$, to check for repetition.
350	The punctuation is removed from the sentence or question and a space is added at the beginning of A$. If your entry does not include closing punctuation, the program cannot function accurately.
360	The program then returns to the main program.
370	This line separates sections of the program.
380-540	This subroutine exchanges words for their synonyms or antonyms.
380	This line identifies this section of the program.
390	If X has a value of zero, then return because there is no entry, just a RETURN character.
400	This line starts an I loop to start reading the antonym/synonym word pairs.
410	This line reads the word pairs. E$ is the word that might be found in the sentence and N$ is the word that will replace it if it is found in the sentence.
420	S is a loop that searches for the spaces between words in the sentence to find the beginnings of possible words to replace.
430	If the character in the sentence is not a space, then the index, S, is incremented to point to the next letter by going to line 530.
440	If a word in the sentence matches one of the words in the DATA statements in lines 450-520, then E$ is replaced by N$ in the sentence.
450	MOM is replaced by MOTHER, DAD is replaced by FATHER, and KIDS is replaced by CHILDREN.
460-520	In these lines the first word of each pair, if found in a sentence, is replaced by the second word of each pair. Markers made of the asterisk or @ sign prevent a word from being replaced twice by the computer. For instance, MYSELF is replaced by YOURSELF* and YOURSELF is replaced by MYSELF. Without the marker on YOURSELF*, MYSELF would be converted to YOURSELF and then back to MYSELF again. Markers prevent this.
530	This is the end of the S and I loops.
540	The program then returns to the calling section.
550	This is a separation line.
560-940	This section actually looks for key words and key phrases.

560	The name of the section of the program.
570	If the user pressed only the RETURN key, the input was not a sentence or question and the program returns for more input.
580	This searches for up to U number of key words or phrases by using an I loop.
590	E$ is the key word or key phrase. J is a number associated with that key word or phrase. J will indicate to the computer what output to print.
600	This searches the entry for the location of possible key words, just like it searched for synonyms above.
620	If a key word is found, R$ is defined and the program branches to line 1910.
630	This line ends the S and I loops.
640	If the program gets through the entire S and I loops, no key words were found in the input. (Line 1910 dealt with keywords if they were found.)
650	R$ is defined and control returns to the calling section.
660–940	This is the list of key words or key phrases and their associated numbers. Notice that COMPUTER and MACHINE have the same number. This number, 1, indicates to the program to print out reply number 1, found in line 1310. Likewise, the other DATA statements contain key words and associated numbers indicating the type of response to display. You can add more of these to include a wider range of subjects. However, the program will respond more slowly. Be sure to change U in line 180 to reflect any change.
945	This is a separation line.
950–1100	This subroutine is used if no keywords were found in the input.
950	This remark identifies this section.
960	X is assigned the value of 1. If MC is zero, control goes to line 980. MC is the stack pointer for statements with the word MY in them. Some of the time when no key words are found, the program will refer back to some of the MY subjects.
970	This line branches to an arbitrary answer if MY was never used.
980	If the user never typed MY, the program will print out one of the lines 990–1020.
990–1020	These are four possible general answers for when no key words were found and when the user never used the word MY. Sometimes these lines will be used even if the word MY was used. You can add non-

committal answers of your own by changing line 980 and adding the appropriate line numbers. If you do this be sure to change the 4 in line 980 to the number of responses that you end up with. Additional replies will not slow the program and will increase the vocabulary of the program.

1030	This line defines Y$ as an arbitrary element of the MY$ array. Thus the computer will try to discuss more about a sentence that had contained the word MY. RAN is a variable that is set to a random number between one and six.
1040	This line branches to one of six possible lines.
1050-1100	These lines print out various sentences and questions using subjects of the MY entries. You can add new responses of your own, and it will not slow down the computer.
1105	This is a separation line.
1110-1160	This subroutine simply eliminates all occurrences of the * and the @ symbols.
1170	This is a separation line.
1180-1280	This subroutine branches to the proper line number to print out the reply based on the value of J, the number associated with the key word. If X was set at 1 earlier, the subroutine simply returns without printing out anything from this subroutine.
1290	This is a separation line.
1300-1750	These are the different replies by the computer. 1310 is printed when J is 1. 1320 is printed when J is 2 and so forth. Some of these lines contain subroutine calls for certain cases.
1760	This is a separation line.
1170-1810	This subroutine is used when the word MY is found in an input. The response is placed in the MY$ stack array. When that array becomes full, arbitrary elements of that array are replaced by the newest entries.
1830-1850	This subroutine actually moves the @ sign from the entries.
1870-1890	This subroutine removes the asterisk from words containing it. Notice that @ is used for E$ with 4 letters and * is used for words with 7 letters. You might think about changing this convention.
1910-1920	This subroutine was used with the key word search. If the key word was found, the program went to line 650 after first going to line 1910.
1960-2050	This subroutine is a set of answers that the computer prints when it encounters the word MY in a

sentence. Notice that there are eight different replies. You can add to these also.

2070 This line is printed if the word HE is found in an input.

2080 This line is printed if the word SHE is found in an input. You might want to add other responses here too. Insert a random number routine as in the other response subroutines.

2090 This line of colons is the end of the program.

Program Operation. After loading or typing the program, into memory, type RUN. The program will clear the screen and type, HELLO. WHAT'S ON YOUR MIND? The computer will use white letters and your input will be in cyan. Be sure that you include a period or question mark at the end of your input. To erase any of your input, simply press DEL to remove letters from right to left. To stop the program, press the RUN/STOP key. Otherwise the program will continue indefinitely.

There are many ways that the program could be changed. Some of those ways have been mentioned in the line descriptions. You might want to expand the program. Maybe you would like to use disk files for storing information that has been input. You might also want to eliminate the rereading of all the DATA statements and the use of the RESTORE command. You might want to store input that contains specific words as was done with the word MY. You might want to correlate some of the responses in other ways.

The Perfect Logic Program

The Perfect Logic, shown in Listing 4-2 is a conversational logic program. Listing 4-3 shows one way of changing the program for use with a disk rather than with a cassette. You type in various statements and questions, complete with final periods and question marks, and the computer will respond accordingly. If you make a statement like ALL CATS ARE ANIMALS, the computer will respond with a question that is the converse, namely, ARE ALL ANIMALS CATS? In this way the computer decides the relationship between the nouns using set theory. If all cats are animals, then cats are contained in the set of animals. Because you would reply NO to the converse question, the computer would conclude that only some animals are cats. (By changing line 10125 in the program, you can have the computer print this out. See the listing and its explanation.) If you replied YES to the converse question, the computer would conclude that all animals are cats. In other words, animals are another name for cats. Such would be the case if you typed, CATS ARE FELINES. The computer would respond, ARE ALL FELINES CATS? You would respond YES.

Listing 4-2 Perfect Logic

```
10 ::::::::::::::::::::::::::::::::::::
20 REM   PERFECT LOGIC - PROGRAM 11
30 REM WRITTEN BY TIMOTHY J. O'MALLEY
40 REM COPYRIGHT 1983, TAB BOOKS INC.
50 REM (WRITTEN FOR THE COMMODORE 64)
60 ::::::::::::::::::::::::::::::::::::
100 DIM SS$(100),NM$(100),AJ$(100),LK%(100,100):ID=1
110 FORI=1TO100:SS$(I)="S":LK%(I,I)=3:NEXTI:PRINTCHR$(147);
115 PRINTTAB(12);"PERFECT LOGIC":PRINT
116 WH$="N":INPUT "DO YOU WANT TO LOAD FROM TAPE (Y/N)";WH$
117 IF WH$="Y" THEN GOSUB 21000
120 PRINT"(LBLU)";:OPEN1,0:INPUT#1,IQ$:PRINT:CLOSE1:PRINT"(CYAN)";
125 IF IQ$="STOP"ORIQ$="END"THENGOSUB 22000:PRINT"(LBLU)":END
130 IP$=LEFT$(IQ$,LEN(IQ$)-1)
140 IFLEFT$(IP$,9)="WHAT ARE "THENGOSUB1000:GOTO1200
145 IFLEFT$(IP$,8)="ARE ALL "THENGOSUB19000:GOTO1200
150 IFLEFT$(IP$,8)="WHAT IS "THENGOSUB2000:GOTO1200
155 IFLEFT$(IP$,7)="WHAT'S "THENGOSUB2500:GOTO1200
160 IFLEFT$(IP$,9)="ARE SOME "THENGOSUB3000:GOTO1200
165 IFLEFT$(IP$,13)="ARE NOT SOME "THENGOSUB3500:GOTO1200
170 IFLEFT$(IP$,4)="ARE "THENGOSUB4000:GOTO1200
173 IF LEFT$(IP$,12)="AREN'T SOME "THENGOSUB4250:GOTO1200
175 IF LEFT$(IP$,7)="AREN'T "THENGOSUB4500:GOTO1200
180 IFLEFT$(IP$,3)="IS "THENGOSUB5000:GOTO1200
185 IF LEFT$(IP$,6)="ISN'T "THENGOSUB5500:GOTO1200
190 IFLEFT$(IP$,7)="IS NOT "THENGOSUB6000:GOTO1200
195 IFLEFT$(IP$,4)="ALL "THENGOSUB19500:GOTO1200
200 IFLEFT$(IP$,5)="SOME "THENGOSUB7000
210 FORI=1TOLEN(IP$):IFMID$(IP$,I,8)=" IS NOT "THENGOSUB8000:GOTO1200
220 IFMID$(IP$,I,7)=" ISN'T "THENGOSUB9000:GOTO1200
230 IFMID$(IP$,I,4)=" IS "THENGOSUB10000:GOTO1200
240 IFMID$(IP$,I,9)=" ARE NOT "THENGOSUB11000:GOTO1200
250 IFMID$(IP$,I,8)=" AREN'T "THENGOSUB12000:GOTO1200
260 IFMID$(IP$,I,5)=" ARE "THENGOSUB13000:GOTO1200
270 NEXTI:GOSUB14000:GOTO1200
999 ::::::::::::::::::::::::::::::::::::
1000 REM UNDERSTAND "WHAT ARE "
1010 LT$=MID$(IP$,10):IFRIGHT$(LT$,1)="S"THENLT$=LEFT$(LT$,LEN(LT$)-1)
     :F1=1
1020 FORI=IDTO1STEP-1:IFNM$(I)=LT$THENJ1=I:GOTO1040
1030 NEXTI:GOSUB18000:RETURN
1040 FORI=IDTO1STEP-1:IFLK%(J1,I)=3ANDLK%(I,J1)=3THENPRINT"ALL ";
1045 IFLK%(J1,I)=3THENPRINTNM$(J1)SS$(J1)" ARE "NM$(I)SS$(I)"."
1050 IFLK%(J1,I)=2THENPRINT"SOME "NM$(J1)SS$(J1)" ARE "NM$(I)SS$(I)"."
1060 IFLK%(J1,I)=1THENPRINT"NO "NM$(J1)SS$(J1)" ARE "NM$(I)SS$(I)"."
1070 NEXTI:RETURN
1199 ::::::::::::::::::::::::::::::::::::
1200 REM COMPUTER DRAWS CONCLUSIONS
1210 FORI1=1TOID-1:FORI2=1TOID-1:IFLK%(I1,I2)=0ORLK%(I2,I1)=0THEN1250
1215 FORI3=1TOID-1:IFLK%(I2,I3)=0ORLK%(I3,I2)=0THEN1240
1217 IFLK%(I1,I3)>0ANDLK%(I3,I1)>0THEN1240
```

```
1220 IFLK%(I1,I2)=3ANDLK%(I2,I1)=3THENGOSUB1300
1225 IFLK%(I1,I2)=3ANDLK%(I2,I1)=2THENGOSUB1400
1230 REM IFLK%(I1,I2)=2ANDLK%(I2,I1)=2THENGOSUB1500
1235 REM IFLK%(I1,I2)=1ANDLK%(I2,I1)=1THENGOSUB1600
1240 NEXTI3
1250 NEXTI2,I1:GOTO120
1300 IFLK%(I2,I3)=3ANDLK%(I3,I2)=3THENLK%(I1,I3)=3:LK%(I3,I1)=3:RETURN
1310 IFLK%(I2,I3)=3ANDLK%(I3,I2)=2THENLK%(I1,I3)=3:LK%(I3,I1)=2:RETURN
1320 IFLK%(I2,I3)=2ANDLK%(I3,I2)=2THENLK%(I1,I3)=2:LK%(I3,I1)=2:RETURN
1330 IFLK%(I2,I3)=1ANDLK%(I3,I2)=1THENLK%(I1,I3)=1:LK%(I3,I1)=1:RETURN
1340 RETURN
1400 REM IFLK%(I2,I3)=3ANDLK%(I3,I2)=3THENLK%(I1,I3)=3:LK%(I3,I1)=2:
     RETURN
1410 IFLK%(I2,I3)=3ANDLK%(I3,I2)=2THENLK%(I1,I3)=3:LK%(I3,I1)=2:RETURN
1420 IFLK%(I2,I3)=1ANDLK%(I3,I2)=1THENLK%(I1,I3)=1:LK%(I3,I1)=1:RETURN
1430 RETURN
1500 REM IFLK%(I2,I3)=3ANDLK%(I3,I2)=3THENLK%(I1,I3)=2:LK%(I3,I1)=2:
     RETURN
1510 REM IFLK%(I2,I3)=3ANDLK%(I3,I2)=2THENLK%(I3,I1)=2:RETURN
1520 REM RETURN
1600 REM IFLK%(I2,I3)=3ANDLK%(I3,I2)=3THENLK%(I1,I3)=1:LK%(I3,I1)=1:
     RETURN
1610 REM RETURN
1999 ::::::::::::::::::::::::::::::::::
2000 REM UNDERSTAND "WHAT IS "
2010 LT$=MID$(IP$,9):IX$=LT$:GOSUB16000:LT$=IX$
2020 FORI=IDTO1STEP-1:IFNM$(I)=LT$THENJ1=I:GOTO1040
2030 NEXTI:GOSUB14000:RETURN
2499 ::::::::::::::::::::::::::::::::::
2500 REM UNDERSTAND "WHAT'S "
2510 IP$=" "+IP$:GOTO2010
2599 :.::::::::::::::::::::::::::::::::::
3000 REM ANSWER "ARE SOME " QUESTION
3010 IP$=MID$(IP$,10):GOTO4020
3499 ::::::::::::::::::::::::::::::::::
3500 REM "ARE NOT SOME " QUESTION
3510 IP$=MID$(IP$,14):GOTO4020
3999 ::::::::::::::::::::::::::::::::::
4000 REM ANSWER "ARE " QUESTION
4010 IP$=MID$(IP$,5)
4020 FORM=1TOLEN(IP$):IFMID$(IP$,M,1)=" "THEN4040
4030 NEXTM:GOTO14000
4040 LT$=LEFT$(IP$,M-1)
4050 IFRIGHT$(LT$,1)="S"THENLT$=LEFT$(LT$,LEN(LT$)-1)
4060 FORJ=IDTO1STEP-1:IFLT$=NM$(J)THEN4080
4070 NEXTJ:GOTO4030
4080 RT$=MID$(IP$,M+1):IX$=RT$:GOSUB16000:RT$=IX$:A2$=AR$
4085 IFRIGHT$(RT$,1)="S"THENRT$=LEFT$(RT$,LEN(RT$)-1)
4090 FORK=IDTO1STEP-1:IFRT$=NM$(K)THEN4110
4100 NEXTK:GOTO4070
4110 IFLK%(J,K)=0THENPRINT"I'M NOT SURE.":RETURN
4120 IFLK%(J,K)=1THENPRINT"NONE ARE.":RETURN
4130 IFLK%(J,K)=2THENPRINT"SOME ARE.":RETURN
```

```
4140 IFLK%(J,K)=3THENPRINT"YES. ALL ARE.":RETURN
4249 ::::::::::::::::::::::::::::::::::
4250 REM "AREN'T SOME " QUESTION
4260 IP$=MID$(IP$,13):GOTO4020
4500 REM "AREN'T ." QUESTION
4510 IP$=MID$(IP$,8):GOTO4020
4999 ::::::::::::::::::::::::::::::::::
5000 REM "IS " QUESTION
5010 IP$=MID$(IP$,4)
5020 FORM=1TOLEN(IP$):IFMID$(IP$,M,1)=" "THEN5040
5030 NEXTM:GOTO14000
5040 LT$=LEFT$(IP$,M-1)
5050 IX$=LT$:GOSUB16000:LT$=IX$:A1$=AR$
5060 FORJ=IDTO1STEP-1:IFLT$=NM$(J)THEN5080
5070 NEXTJ:GOTO5030
5080 RT$=MID$(IP$,M+1):IX$=RT$:GOSUB16000:RT$=IX$:A2$=AR$
5090 FORK=IDTO1STEP-1:IFRT$=NM$(K)THEN5110
5100 NEXTK:GOTO5070
5110 GOTO4110
5499 ::::::::::::::::::::::::::::::::::
5500 REM "ISN'T " QUESTION
5510 IP$=MID$(IP$,7):GOTO5020
5599 ::::::::::::::::::::::::::::::::::
6000 REM "IS NOT " QUESTION
6010 IP$=MID$(IP$,8)GOTO5020
6999 ::::::::::::::::::::::::::::::::::
7000 REM "SOME " STATEMENT
7010 IP$=MID$(IP$,6):FS=1:RETURN
7999 :::::::::::::::::::::::::::::::::::
8000 REM UNDERSTAND " IS NOT "
8010 LT$=LEFT$(IP$,I-1):RT$=MID$(IP$,I+8)
8020 IF RIGHT$(IQ$,1)="?"THENGOSUB15000:RETURN
8030 GOTO 12020
9010 LT$=LEFT$(IP$,I-1):RT$=MID$(IP$,I+7)
9020 IF RIGHT$(IQ$,1)="?"THENGOSUB15000:RETURN
9030 GOTO 12020
9999 ::::::::::::::::::::::::::::::::::
10000 REM UNDERSTAND IS SENTENCE
10010 LT$=LEFT$(IP$,I-1):RT$=MID$(IP$,I+4)
10030 IX$=LT$:GOSUB16000:LT$=IX$:A1$=AR$
10040 IX$=RT$:GOSUB16000:RT$=IX$:A2$=AR$
10045 IF RIGHT$(IQ$,1)="?"THENGOSUB15000:RETURN
10050 FORI=IDTO1STEP-1:IFLT$=NM$(I)THENJ1=I:GOTO10070
10060 NEXTI:NM$(ID)=LT$:AJ$(ID)=A1$:J1=ID:ID=ID+1
10070 FORI=IDTO1STEP-1:IFRT$=NM$(I)THENK1=I:GOTO10085
10080 NEXTI:NM$(ID)=RT$:AJ$(ID)=A2$:K1=ID:ID=ID+1
10085 IF LK%(J1,K1)=3 THEN ON 1+3*RND(1) GOTO 10087,10088,10089
10086 GOTO 10090
10087 PRINT"I KNEW THAT ALREADY.":RETURN
10088 PRINT"I HEARD THAT BEFORE.":RETURN
10089 PRINT"I KNOW.":RETURN
10090 LK%(J1,K1)=3
10095 IFFSTHENFS=0:LK%(J1,K1)=2
```

```
10097 IF LK%(K1,J1)>0 THEN GOSUB 17000:RETURN
10100 PRINT"ARE ALL "NM$(K1)SS$(K1)" "NM$(J1)SS$(J1)"?"
10110 PRINT"(LBLU)";:OPEN1,0:INPUT#1,AN$:PRINT:CLOSE1:PRINT"(CYAN)";
10120 IFLEFT$(AN$,1)="Y"THENLK%(K1,J1)=3:GOSUB17000:RETURN
10125 IF LEFT$(AN$,1)="N" THEN LK%(K1,J1)=2:GOSUB 17000:RETURN
10130 PRINT"SOME "NM$(K1)SS$(K1)" ARE "NM$(J1)SS$(J1)". ":GOSUB17000:
      RETURN
10999 :::::::::::::::::::::::::::::::::
11000 REM UNDERSTAND " ARE NOT "
11010 LT$=LEFT$(IP$,I-1):RT$=MID$(IP$,I+9):GOTO12020
11999 :::::::::::::::::::::::::::::::
12000 REM UNDERSTAND " AREN'T "
12010 LT$=LEFT$(IP$,I-1):RT$=MID$(IP$,I+8)
12020 IX$=LT$:GOSUB16000:LT$=IX$:A1$=AR$
12030 IX$=RT$:GOSUB16000:RT$=IX$:A2$=AR$
12040 IFRIGHT$(LT$,1)="S"THENLT$=LEFT$(LT$,LEN(LT$)-1):F1=1
12050 IFRIGHT$(RT$,1)="S"THENRT$=LEFT$(RT$,LEN(RT$)-1):F2=1
12060 FORI=IDTO1STEP-1:IFLT$=NM$(I)THENJ1=I:GOTO12080
12070 NEXTI:NM$(ID)=LT$:AJ$(ID)=A1$:J1=ID:ID=ID+1
12080 FORI=IDTO1STEP-1:IFRT$=NM$(I)THENK1=I:GOTO12100
12090 NEXTI:NM$(ID)=RT$:AJ$(ID)=A2$:K1=ID:ID=ID+1
12100 IFF1THENSS$(J1)="S":F1=0
12110 IFF2THENSS$(K1)="S":F2=0
12120 GOTO20000
12999 ::::::::::::::::::::::::::::::::
13000 REM UNDERSTAND " ARE "
13010 LT$=LEFT$(IP$,I-1):RT$=MID$(IP$,I+5)
13020 IX$=LT$:GOSUB16000:LT$=IX$:A1$=AR$
13030 IX$=RT$:GOSUB16000:RT$=IX$:A2$=AR$
13040 IFRIGHT$(LT$,1)="S"THENLT$=LEFT$(LT$,LEN(LT$)-1):F1=1
13050 IFRIGHT$(RT$,1)="S"THENRT$=LEFT$(RT$,LEN(RT$)-1):F2=1
13060 FORI=IDTO1STEP-1:IFLT$=NM$(I)THENJ1=I:GOTO13080
13070 NEXTI:NM$(ID)=LT$:AJ$(ID)=A1$:J1=ID:ID=ID+1
13080 FORI=IDTO1STEP-1:IFRT$=NM$(I)THENK1=I:GOTO13100
13090 NEXTI:NM$(ID)=RT$:AJ$(ID)=A2$:K1=ID:ID=ID+1
13100 SS$(J1)="":IFF1THENSS$(J1)="S":F1=0
13110 SS$(K1)="":IFF2THENSS$(K1)="S":F2=0
13120 GOTO10085
13999 ::::::::::::::::::::::::::::::::
14000 REM DO NOT UNDERSTAND INPUT
14005 PRINTLT$;"? ";
14010 ON1+INT(8*RND(1))GOTO14020,14030,14040,14050,14060,14070,14080,
      14090
14020 PRINT"I DON'T DIG.":RETURN
14030 PRINT"WHAT?":RETURN
14040 PRINT"WHAT ARE YOU SAYING?":RETURN
14050 PRINT"HUH?":RETURN
14060 PRINT"I DON'T FOLLOW.":RETURN
14070 PRINT"I DON'T UNDERSTAND.":RETURN
14080 PRINT"YOU ARE NOT MAKING SENSE.":RETURN
14090 PRINT"IF YOU SAY SO, BUT I DON'T KNOW.":RETURN
14999 ::::::::::::::::::::::::::::::::
15000 REM STATEMENT IS REALLY QUESTION
```

```
15010 FORI=IDTO1STEP-1:IFLT$=NM$(I)THEN15030
15020 NEXTI:GOTO14000
15030 FORK=IDTO1STEP-1:IFRT$=NM$(K)THEN15050
15040 NEXTK:GOTO14000
15050 GOTO5110
15999 :::::::::::::::::::::::::::::::::
16000 REM FIND ARTICLES OF IX$
16010 DATA " THE "," A "," AN ","THE ","A ","AN "
16020 AR$="":FORI=1TO6:READAT$:LA=LEN(AT$)
16030 IFMID$(IX$,1,LA)=AT$THENIX$=MID$(IX$,LA+1):AR$=AT$:RESTORE:
      RETURN
16040 NEXTI:RESTORE:RETURN
16999 ::::::::::::::::::::::::::::::::::
17000 REM INPUT UNDERSTOOD
17010 ON1+INT(8*RND(1))GOTO17020,17030,17040,17050,17060,17070,17080,
      17090
17020 PRINT"I UNDERSTAND NOW.":RETURN
17030 PRINT"OKAY.":RETURN
17040 PRINT"ALL RIGHT.":RETURN
17050 PRINT"I SEE.":RETURN
17060 PRINT"OK. THAT MAKES SENSE.":RETURN
17070 PRINT"I GET IT. WHAT ELSE CAN I LEARN?":RETURN
17080 PRINT"NOW WE ARE GETTING SOMEWHERE.":RETURN
17090 PRINT"GOOD. LET'S GO ON.":RETURN
17999 ::::::::::::::::::::::::::::::::::
18000 REM DID NOT FIND NOUN IN QUESTION
18010 ON1+INT(RND(1))GOTO18020,18030,18040,18050,18060,18070,18080,
      18090
18020 PRINT"YOU'RE ASKING ME? I DON'T KNOW ABOUT "LT$;:GOTO18100
18030 PRINT"GOT ME. I HAVE NO INFO ON "LT$;:GOTO18100
18040 PRINT"BEATS ME.":RETURN
18050 PRINT"I WOULD NOT KNOW ABOUT "LT$;:GOTO18100
18060 PRINT"I HAVE NO INFO ABOUT "LT$;:GOTO18100
18070 PRINT"YOUR GUESS IS AS GOOD AS MINE.":RETURN
18080 PRINT"GOOD QUESTION. I DON'T KNOW.":RETURN
18090 PRINT"I HAVE NO IDEA.":RETURN
18100 IF F1THENPRINT"S";:F1=0
18110 PRINT".":RETURN
18999 :::::::::::::::::::::::::::::::::
19000 REM "ARE ALL " QUESTION
19010 IP$=MID$(IP$,9):GOTO4020
19499 :::::::::::::::::::::::::::::::::::
19500 REM "ALL " STATEMENT
19510 IP$=MID$(IP$,5):RETURN
19999 ::::::::::::::::::::::::::::::::::
20000 REM SET NOT CONDITION
20010 LK%(J1,K1)=1:LK%(K1,J1)=1:GOTO17000
20998 :::::::::::::::::::::::::::::::::
20999 REM    *** LOAD DATA ***
21000 D$="":INPUT"WHAT FILE NAME";D$
21010 OPEN1,1,0,D$
21020 INPUT#1,ID
21030 FOR I=1 TO ID-1:INPUT#1,SS$(I):NEXT I
```

```
21040 FOR I=1 TO ID-1:INPUT#1,NM$(I):NEXT I
21050 FOR I=1 TO ID-1:INPUT#1,AJ$(I):NEXT I
21060 FOR I=1 TO ID-1:FORJ=1TOID-1:INPUT#1,LK:LK%(I,J)=LK:NEXT J,I
21070 CLOSE 1:RETURN
21998 ::::::::::::::::::::::::::::::::::
21999 REM    *** SAVE DATA ***
22000 WH$="N":INPUT"WANT TO SAVE WORK (Y/N)";WH$:IFWH$="N"THEN END
22005 D$="":INPUT "WHAT FILE NAME";D$
22010 OPEN1,1,1,D$
22020 PRINT#1,ID
22030 FOR I=1 TO ID-1:PRINT#1,SS$(I):NEXT I
22040 FOR I=1 TO ID-1:PRINT#1,NM$(I):NEXT I
22050 FOR I=1 TO ID-1:PRINT#1,AJ$(I):NEXT I
22060 FOR I=1 TO ID-1:FORJ=1TOID-1:LK=LK%(I,J):PRINT#1,LK:NEXT J,I
22070 CLOSE 1:RETURN
22999 ::::::::::::::::::::::::::::::::::
```

Listing 4-3 Changes To Use Perfect Logic on Disk

```
116 WH$="N":INPUT "DO YOU WANT TO LOAD FROM DISK (Y/N)";WH$

20999 REM    *** LOAD DATA ***
21000 D$="":INPUT"WHAT FILE NAME";D$
21005 DF$=D$:DF=1
21010 OPEN 2,8,2,"0:"+D$+",S,R"
21020 INPUT#2,ID
21030 FOR I=1 TO ID-1:INPUT#2,SS$(I):NEXT I
21040 FOR I=1 TO ID-1:INPUT#2,NM$(I):NEXT I
21050 FOR I=1 TO ID-1:INPUT#2,AJ$(I):NEXT I
21060 FOR I=1 TO ID-1:FORJ=1TOID-1:INPUT#2,LK:LK%(I,J)=LK:NEXT J,I
21070 CLOSE 2:RETURN

21998 ::::::::::::::::::::::::::::::::::
21999 REM    *** SAVE DATA ***
22000 WH$="N":INPUT"WANT TO SAVE WORK (Y/N)";WH$:IFWH$="N"THEN END
22001 IF DF=0 THEN 22013
22002 PRINT"DO YOU WANT TO SAVE THIS FILE UNDER THE SAME FILENAME THAT

22003 INPUT "JUST USED";SN$
22004 IFLEFT$(SN$,1)<>"Y" THEN 22013
22005 D$=DF$:OPEN 15,8,15:PRINT#15,"S0:"+D$:CLOSE 15:GOTO 22018
22013 PRINT"█DO NOT USE THE NAME OF A FILE THAT IS   ALREADY ON THE
      DISK.█"
22015 D$="":INPUT "WHAT FILE NAME";D$
22016 IF D$="" THEN 22015
22017 DF$=D$:DF=1
22018 OPEN 2,8,2,"0:"+D$+",S,W"
22020 PRINT#2,ID
22030 FOR I=1 TO ID-1:PRINT#2,SS$(I):NEXT I
22040 FOR I=1 TO ID-1:PRINT#2,NM$(I):NEXT I
22050 FOR I=1 TO ID-1:PRINT#2,AJ$(I):NEXT I
22060 FOR I=1 TO ID-1:FORJ=1TOID-1:LK=LK%(I,J):PRINT#2,LK:NEXT J,I
22070 CLOSE 2:RETURN
```

That way CATS and FELINES are sets that contain all of each other.

You can type statements like PLANTS ARE NOT ANIMALS. The computer would immediately conclude that ANIMALS ARE NOT PLANTS. By using logic, the computer also concludes that every noun (or set) is a subset of itself. For example, if you type, ARE ALL CATS CATS? The computer would respond in the affirmative. The computer draws various conclusions based on set theory. If all cats are animals and animals are not plants, the computer concludes that NO CATS ARE PLANTS, and NO PLANTS ARE CATS.

Figure 4-2 is an explanation of the kinds of conclusions that the computer will draw. The computer will make comparisons between three sets. If certain conditions are met, the computer will make changes in LK%, as results. Zeros in LK% represent an absence of information. In that case, the computer doesn't know about the relationship between two sets. If the relationship between two sets is a 1, then the sets are not related. In fact, they may be opposites, as in the case of plants and animals. If the

CONDITIONS				RESULTS	
LK%(13,11)	LK%(11,12)	LK%(12,11)	LK%(12,13)	LK%(13,12)	LK%(11,13)
3	3	3	3	3	3
2	3	3	3	2	3
2	3	3	2	2	2
1	3	3	1	1	1
2	3	2	3	2	3
1	3	2	1	1	1

LEGEND:
1 = Sets are not related and may be opposites.
2 = Some of one set are contained in the second set.
3 = All of one set is contained in the second set.

LK%(I1,I2) is the relationship between I1 and I2.
LK%(I2,I1) is the relationship between I2 and I1.
LK%(I2,I3) is the relationship between I2 and I3.
LK%(I3,I2) is the relationship between I3 and I2.
LK%(I1,I3) is the relationship between I1 and I3.
LK%(I3,I1) is the relationship between I3 and I1.
I1 is the first set.
I2 is the second set.
I3 is the third set.

Fig. 4-2. The logic used in the Perfect Logic program.

relationship between the first set and the second is a 2, then some, but not all, of the first set is contained in the second set. If the relationship between the first set and the second set is a 3, then all of the first set is contained in the second set. The ? in the figure indicates an indeterminant result. The question mark is either 2 or 3. Thus the program scans the LK% array looking for elements that meet one of the six conditions in Fig. 4-2. When it finds one, it makes changes to other elements of the array. In this way, the program really demonstrates artificial intelligence. (We never really told the computer that a cat was not a plant. It concluded it from the statements that said a cat was an animal and that animals are not plants.)

This program shows some degree of self-correction if you delete line 10097. It might even correct some things if you don't. The maximum number of items that the program can store in its present form is 100. That is because the LK% array is 100 × 100 elements. This is 10,000 elements, a big bite of memory (no pun intended). This program slows down as the number of sets increases. That is because the search procedure takes longer and longer as the program tries to draw conclusions. You might think of ways to get around this or think of ways to store this information on disk instead of in RAM. It may be somewhat difficult because the number of possible relationships between sets is the square of the number of sets. Maybe you would adopt a different numbering convention to designate the kinds of sets involved. There are other ways to change the program also.

Here's a line-by-line description of the program.

LINE	**EXPLANATION**
10–60	These lines provide identification for the program.
100	This line dimensions the various arrays used in the program. SS$ is an array used to store the plural of the nouns entered. NM$ is the array that stores the noun of the phrase as determined by the program. AJ$ stores the articles (a or an) used with the nouns. LK%. is the array that contains the code (0–3) for determining the relationships between nouns or noun phrases entered. LK% is the most important array in the program. ID is a variable that acts as an index. It is set at 1 at the start of the program.
110	This line sets all of the plural forms of nouns that it encounters, making them end with S. The diagonal of the LK% array is set as 3, meaning that everything is a set that contains itself. CHR$(147) is the code used to clear the screen.

115	This line tabs across and prints the first line on the screen.
116	This line asks if you want to load data from cassette tape. You enter Y or N.
117	If you enter Y, the program calls a subroutine that starts at line 21000.
120	The computer changes the color to light blue (ASCII code 154). This line then asks for an imput by opening the keyboard file. After the input, the computer prints in the cyan color, the computer's color.
125	If your input was STOP or END, the program branches to the subroutine that starts at line 22000. That subroutine asks if you want to save the data accumulated during the session on cassette tape for use with the next session. This way the computer "remembers" what it had learned. Finally, the color is changed back to light blue, and the program ends.
130	IP$ is IQ$ without the last letter. We drop the punctuation—if you forgot to enter the punctuation with your input, the program will not function accurately.
140	If the first 9 letters that you typed are WHAT ARE, the program calls the subroutine that deals with "WHAT ARE" questions, located at line 1000. Then the program branches to line 1200.
145	Likewise if a question starts with ARE ALL, the program goes to a subroutine that starts at line 19000 and then goes to line 1200. Line 1200 is a routine that attempts to draw conclusions from that data entered.
150-200	These lines do similar things with input that starts with various words, such as WHAT IS , IS , and SOME . These are the only kinds of things that the program can deal with. The program will attempt to look for a noun or nouns following these words. Then it will respond accordingly.
210-270	These lines are a loop that attempts to locate the verbs IS, IS NOT, ISN'T, ARE, ARE NOT and AREN'T if the lines above did not locate any starting words in the input. If the loop runs through its entirety without finding any of these verbs, the program goes to the subroutine that starts at line 14000, where it prints out a message saying it didn't understand the input. Then the program branches to line 1200 to see if it can draw some more conclusions before returning for more input.
999	This line separates the main program from the next section, the WHAT ARE subroutine.

1000–1070	This subroutine attempts to interpret questions beginning with WHAT ARE . . .
1010	LT$ is the input starting with the tenth character. If the last word ends with an S, it is assumed that it is a plural form, and the S is dropped from LT$. A "flag" F1 is set at 1.
1020	This line searches for the noun starting with the most recently entered character and working backwards. If it matches the nouns stored in the NM$ array to the input, J1 is set to the index, I, and the program branches to line 1040.
1030	If the program goes through the entire NM$ array without finding the array, the program calls the subroutine that starts at line 18000 and then returns for more input.
1040–1070	This I loop prints out all the sets to which the noun belongs. What is printed depends on the contents of LK%. NM$(J1) is the noun. SS$(J1) is its plural form.
1199	This is a separating line.
1200–1610	This is the most important subroutine; it is used to draw conclusions. It makes comparisons between three sets. For example if set A contains all of set B and set C contains all of set A, it can conclude that all of set B are contained by set C and that some of set C is set B. Likewise if set A contains all of set B and set C contains none of set A, it can conclude that C contains none of set B. All of the logical combinations that yield definite conclusions are included in the subroutine. In the example that has been mentioned, A might be ANIMALS, B might be CATS, and C might be PLANTS. If we know that all CATS are ANIMALS and no ANIMALS are PLANTS, it follows that no CATS are PLANTS. (Figure 4-2 shows all of these combinations of conditions and the results.)
1200	This line identifies the subroutine.
1210	I1 is the index for one of three nested loops that examines entries in LK%. I2 is the index for the second set. If the relationship between set I1 and set I2 is 0 (both ways), nothing can be concluded, and the program branches to line 1250 to increase the indexes of the loops.
1215	I3 is the index for the third set. If I2 and I3 have no relationship at all, nothing can be concluded. The program then branches to line 1240 to increase the I3 loop index.

1217	If the relationship between I1 and I3 and the relationship between I3 and I1 have already been established, control goes to line 1240.
1220	If I1 is entirely contained in I2, and I2 is entirely contained in I1, then I1 is the same as I2, and control goes to the subroutine at line 1300 where the relationships between I2 and I3 are checked.
1225	If I1 is a proper subset of I2, control goes to line 1400 where the relationships between I2 and I3 are tested.
1230	If some of I1 is contained in I2 and some of I2 is contained in I1, control goes to line 1500 where the relationships between I2 and I3 are tested.
1235	If there is definitely no relationships between I1 and jI1 (I1 and I2 may be opposites), there is a call to the subroutine at line 1600.
1240	This line increments the I3 index (loop).
1250	This line contains the ends of the I2 and I1 loops and a branch to line 120 for more input.
1300-1340	This subroutine tests and sets the results of various elements of the LK% array as called by line 1220.
1400-1430	This subroutine tests and sets the various elements of the LK% array as called by line 1225.
1500-1520	This subroutine tests and sets the various elements of the LK% array as called by line 1230.
1600-1610	This subroutine tests and sets the final possible elements of LK%.
1999	This is a separation line.
2000-2030	This subroutine interprets questions that start with WHAT IS . . . LT$ is IP$ from the ninth element to the last element of that string. IX$ is defined as LT$, and then the program calls the subroutine that starts at line 16000. That subroutine looks for the words THE, A or AN in IX$. If it finds any of these words, it drops them from the string. LT$ is then defined as the changed string. Line 2020 does a search for the noun from the modified string. If it finds the noun, it calls part of the WHAT ARE subroutine (starting at line 1040) to print out all the things that the noun is. Line 2030 is the end of the I loop. If the noun is not found, the program branches to line 14000 to indicate that the computer didn't understand the input. You will notice that if the noun was found in line 2020, that the computer branched out of the I loop. Branching out of loops generally causes no serious problems on the Commodore 64, although on some computers it might.
2499	This is a separation line.

2500–2510	In this subroutine, the computer deals with questions that start with WHAT'S . . . To answer these questions, it adds a space at the beginning of IP$ and branches to the WHAT IS subroutine. That subroutine will drop the first 8 characters and process the rest as if the question was WHAT IS.
2599	This is a separation line.
3000–3010	This subroutine processes ARE SOME questions by dropping the first nine characters and branching to the ARE question subroutine that starts at line 4020.
3499	This is a separation line.
3500–3510	This subroutine processes ARE NOT SOME questions by dropping the first 13 characters and branching to line 4020.
3999	This is a separation line.
4000–4140	This subroutine processes all the ARE . . . questions.
4000	This is the remark identifying the subroutine.
4010	This line drops the first four characters of ARE questions.
4020–4030	The M loop searches for the separation between words. If it finds a space, it branches to line 4040. If it does not find anything, it BRANCHES to the subroutine that starts at line 14000. Notice that subroutines can be branched to as well as called if you are careful.
4040	LT$ is the left part of IP$.
4050	This line looks for the letter S, indicating the plural form of the noun. If it finds it, the S is dropped.
4060–4070	These lines check to see if the first word of this string is found in the NM$ array. If it is, the program branches to line 4080 to find the next noun. If the noun is not found, maybe the noun is really a phrase, and the program will use more than one word in LT$. It does this by branching back into the M loop at line 4030.
4080	RT$ is the right part of the original IP$. The rest of this line looks for the articles. THE, A and AN in RT$.
4085	This line drops the S (if it is there) from the end of the last word because it assumes that it is dealing with the plural form of the noun.
4090–4100	This K loop attempts to search for the noun in question. If it finds it, the program branches to line 4110. If it does not find it, it assumes that it made a mistake in determining what LT$ was and will branch back into the J loop! If it does not find it then, it branches

	back to the M loop! If it still finds nothing, it branches to line 14000.
4110–4140	Four different messages will be printed depending on the value of LK%(J,K). J and K are the loop indicators from the lines above. After the message is printed, the program returns to the input subroutine.
4249	This is a separation line.
4250–4260	This subroutine answers the AREN'T SOME question by dropping the first 12 characters and branching into the ARE subroutine.
4500–4510	This subroutine answers the AREN'T question by dropping the first seven characters and branching to line 4020.
4999	This is a separation line.
5000–5110	This subroutine interprets the IS question. It behaves very much like the ARE question subroutine and branches into that subroutine if it finds both nouns or phrases. The number 5110 in line 5090 could be changed to 4110 and line 5110 could be deleted.
5499	This is a separation line.
5500–5510	This subroutine answers the ISN'T question by dropping the first six characters and branching into the IS subroutine.
5599	This is a separation line.
6000–6010	This subroutine answers the IS NOT question by dropping the first seven characters and branching into the IS subroutine.
6999	This is a separation line.
7000–7010	This subroutine is used for statements that begin with SOME. The first five letters are dropped and FS, a flag, is set at 1; then the program returns to the calling routine.
8000–9030	These lines deal with statements and questions that contain the words IS NOT or ISN'T somewhere in the middle of the input. The words are dropped from the input and the two parts are placed in LT$ and RT$. If the input ends with the question mark, the program calls the subroutine starting at line 15000 then returns. For declarative statements, the program branches to line 12020 of the AREN'T sentence subroutine.
9999	This is a separation line.
10000–10130	This subroutine interprets the sentence containing IS. This subroutine will print one of three sentences if the program has already stored or concluded that

	statement beforehand. Lines 10087-10089 are those responses. Line 10100 asks a question to determine whether or not the converse of the statement is true. Line 10110 changes the color of the input and requests a Yes or No answer. The program then changes the color back to cyan. If you delete from line 10125:GOSUB 17000:RETURN the program will acknowledge that sometimes the converse is true when answering No.
10999	This is a separation line.
11000-11010	This subroutine interprets statements with ARE NOT in the middle of the sentence. Then the program branches into the AREN'T subroutine.
11999	This is a separation line.
12000-12120	This subroutine interprets any sentence containing AREN'T. It is like the IS sentence subroutine. It then branches to line 20000, which sets both sets to 1, the not condition. The not condition means that the nouns may be opposites.
12999	This is a separation line.
13000-13120	This subroutine interprets sentences containing ARE. It resembles the AREN'T sentence subroutine. It doesn't set the not condition. It branches into the IS sentence subroutine at line 10085.
13999	This is a separation line.
14000-14090	If the program doesn't understand the input, it calls this subroutine. This subroutine prints up to eight different comments. You might want to change or add some comments to suit your preference.
14999	This is a separation line.
15000-15050	This subroutine is used if the statement entered is really a question because it has a question mark at the end.
15999	This is a separation line.
16000-16040	This is the subroutine that searches for the articles THE, A, or AN in the input. Some of the articles have a space in front of them (if they are in the middle of the input).
16999	This is a separation line.
17000-17090	This subroutine includes eight comments telling the user that the computer has understood the input. You may want to add to or change some of these comments as well.
17999	This is a separation line.
18000-18110	This subroutine is used if the user asks a WHAT IS or WHAT ARE question and the computer has never heard of the noun in question. You might want to

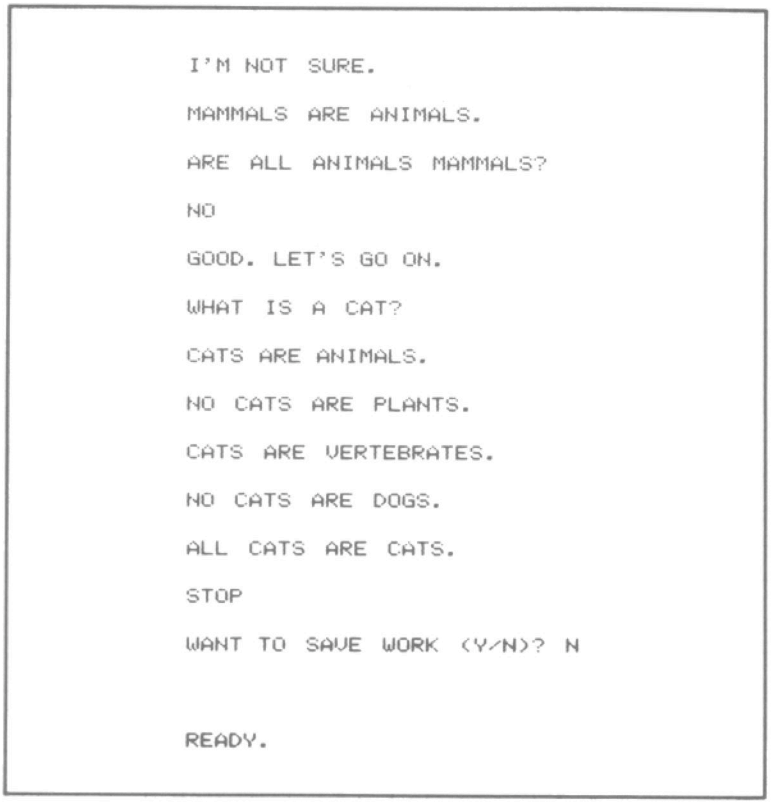
	change some of these comments. Some of the comments print out the noun, LT$, that it can't find. Line 18100 prints S on the end of the word if the noun is plural.
18999	This is a separation line.
19000-19010	This subroutine drops ARE ALL from the input and branches into the ARE question subroutine.
19499	This is a separation line.
19500-19510	This subroutine drops the ALL from the beginning of a statement if it has been included in the input.
19999	This is a separation line.
20000-20010	This sets the not condition for nouns or phrases that are opposites or have a negative relationship.
20998	This is a separation line.
20999-21070	This subroutine allows you to load data that you saved on cassette tape from a former session. You can specify a specific filename or simply press the RETURN key and the name will be an empty string. If you have a disk drive, you might want to alter the numbers to save on disk. This subroutine, then, allows you to accumulate information that the program has handled.
21998	This is a separation line.
21999-22070	This subroutine is the opposite of the last one. It allows you to save your work on cassette tape for use in the next session. You may define the filename or leave it blank. If it is blank, the first file that is found will be loaded. You might want to alter this subroutine if you have a disk drive.
22999	This is the last line of the program!

Program Operation. When you have entered the program into memory, type RUN. The computer will print PERFECT LOGIC at the top of the screen. It will then ask you if you want to load data from tape. Type Y for Yes or N for No. When the cursor blinks, simply enter your statements or questions. All inputs should contain at least one of the following words: IS, ARE, NOT, AREN'T, and ISN'T. You can include other words such as SOME, ALL, WHAT IS, WHAT'S, and WHAT ARE.

Figures 4-3 and 4-4 show typical conversations with the logic program. Every other line is printed by the computer. If you type: A CAT IS A MAMMAL. The computer will respond: ARE ALL MAMMALS CATS? You reply NO. It might then reply: OKAY. You might then type: A DOG IS NOT A CAT. It might reply: I UNDERSTAND NOW. You might type: A MAMMAL IS A VERTEBRATE. It responds: ARE ALL VERTEBRATES MAMMALS? You reply, NO. It prints: NOW WE ARE GETTING

```
DO YOU WANT TO LOAD FROM TAPE (Y/N)? N

A CAT IS A MAMMAL.

ARE ALL MAMMALS CATS?

NO

OKAY.

A DOG IS NOT A CAT.

I UNDERSTAND NOW.

A MAMMAL IS A VERTEBRATE.

ARE ALL VERTEBRATES MAMMALS?

NO

NOW WE ARE GETTING SOMEWHERE.

A CAT IS A MAMMAL.

I HEARD THAT BEFORE.

IS A DOG A MAMMAL?

I'M NOT SURE.

A DOG IS A MAMMAL.

ARE ALL MAMMALS DOGS?

NO

GOOD. LET'S GO ON.

A CAT IS NOT A DOG?

A CAT? I DON'T FOLLOW.

AREN'T CATS DOGS?

NONE ARE.

PLANTS ARE NOT ANIMALS.

OKAY.

ARE SOME PLANTS ANIMALS?

NONE ARE.

ARE SOME ANIMALS CATS?
```

Fig. 4-3. The start of the conversation with the Perfect Logic program.



```
I'M NOT SURE.

MAMMALS ARE ANIMALS.

ARE ALL ANIMALS MAMMALS?

NO

GOOD. LET'S GO ON.

WHAT IS A CAT?

CATS ARE ANIMALS.

NO CATS ARE PLANTS.

CATS ARE VERTEBRATES.

NO CATS ARE DOGS.

ALL CATS ARE CATS.

STOP

WANT TO SAVE WORK (Y/N)? N


READY.
```

Fig. 4-4. The end of the conversation with the Perfect Logic program.

SOMEWHERE. You type: A CAT IS A MAMMAL. It tells you: I HEARD THAT BEFORE. The conversation continues on and on. If you ask: WHAT IS A CAT? It replies: CATS ARE ANIMALS. NO CATS ARE PLANTS. CATS ARE VERTEBRATES. NO CATS ARE DOGS. ALL CATS ARE CATS. You see some of the conclusions that the computer drew from statements entered. You never said that a cat was not a plant.

Except for the limited storage problem and the slow response with a large number of sets, this program demonstrates artificial intelligence well when it makes deductions based on statements using sets. This program utilizes natural language processing. You simply typed in your IS and ARE and NOT statements in English. You asked questions to retrieve information by typing them in English. When the computer doesn't understand what you are typing, it will tell you so.

Chapter 5

Heuristics

In this chapter the discussion of heuristics is continued, and two BASIC programs are examined. The first is a tic tac toe program that remembers strategy, and the second is a five-in-a-row game that you can play with other people and with the computer.

Many of the programs that we have discussed so far have used heuristics. Let's define heuristics and examine some types of heuristics.

A heuristic, in the broadest sense, is a rule of thumb used in solving a problem. It is a way to provide a link between a question and an answer. Heuristics eliminate some of the nonsensical paths in the search for an answer. Heuristics can be generalizations about sets of things and their relationships to each other. Heuristics, then, are any methods that eliminates the searching of the full range of possibilities for an answer.

A pruned search is an example of a heuristic. In the pruned search the possible choices that lead to an immediate failure are deleted, and consequently the number of possible choices are reduced. The binary solution that was used in the Towers of Hanoi program was a heuristic that yielded the solution directly. If a computer game stores moves that lead to a win, it might search the moves for a solution. The program would be learning.

HEURISTIC PROGRAMS IN BASIC

In this chapter I will use heuristics in two BASIC programs. In the tic tac toe program, I will have the computer store moves that lead to a victory. It will also store the last move that lead to a defeat. In this way the computer can avoid or choose a particular move. You won't be able to fool the computer twice with the same set of moves.

In the five-in-a-row game the computer plays against one or several human opponents. The computer will try to get five dots in a row. It will try to block the moves of any player that has three of four dots in a five dot space on the board. If one end of a five dot space is blocked, it will try to form a line of five using the other end. If that is also blocked, it will abandon that section and find a new space. The program does not, however, store strategy.

Let's now look in depth at tic tac toe and five-in-a-row.

Heuristic Tic Tac Toe

The program shown in Listing 5-1 plays the simple game of tic tac toe with you. However, the computer will store all moves that lead to a victory. It also stores the last bad move that lead to a defeat and will avoid using that move again. In short, the program remembers the best moves and will make them when playing the game.

The game is played on a three by three grid. The object is to place three of your symbols in a row either vertically, horizontally, or diagonally on the board. This program displays the move number and the game number. There is no provision to store the strategy for future games. Thus the computer must relearn the strategy each time that the game is loaded and run. You might think about a way to save the strategy on tape or disk.

Here's a line-by-line description of the program.

Listing 5-1 Tic Tac Toe

```
5 ::::::::::::::::::::::::::::::::::::::::
10 REM HEURISTIC TIC TAC TOE - PROG. 12
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
45 ::::::::::::::::::::::::::::::::::::::::
47 REM        *** MAIN PROGRAM ***
50 POKE 51,0:POKE 52,64:POKE 55,0:POKE 56,64
55 PRINT "HANG ON A MINUTE WHILE I DO SOME HOUSE-"
60 PRINT "KEEPING.....THANKS!!"
70 G=0:FOR I=16384 TO 36077:POKE I,0:NEXT
80 G=G+1:T=0
90 PRINT CHR$(147);"      *** HEURISTIC TIC TAC TOE ***":PRINT
100 PRINT "GAME NO. ";G:PRINT
```

```
110 FOR J=1 TO 3
120 FOR K=1 TO 3
130 B(J,K)=0
140 L=3*J+K-3
150 R(L)=0
160 C(L)=0:S(L)=0
170 NEXT K,J
180 M=0:REM M IS THE MOVE NUMBER
190 GOSUB 900:GOSUB 4000:REM DETERMINE WHO GOES FIRST & PRINT BOARD
200 M=M+1:REM INCREASE MOVE NUMBER
210 PRINT:PRINT "MOVE NO. ";M:PRINT
220 M2=M/2:M3=M2<>INT(M2)
230 M4=M2=INT(M2)
240 IF (Z=1 AND M3) OR (Z=2 AND M4) THEN GOSUB 2000:REM COMPUTER'S MOVE
250 IF (Q=1 AND M3) OR (Q=2 AND M4) THEN GOSUB 3000:REM YOUR MOVE
260 GOSUB 4000:REM PRINT BOARD
270 GOSUB 1000:REM ANY WINNER YET?
280 IF A=1 THEN 330:REM STORE WINNING STRATEGY
290 IF M<9 THEN 200
300 PRINT:INPUT "NOBODY WON. WANT TO PLAY AGAIN";R1$
310 IF LEFT$(R1$,1)="Y" THEN 80:REM GO TO THE BEGINNING OF THE NEXT
    GAME
320 END
325 ::::::::::::::::::::::::::::::::::::
327 REM *** STORE WINNING STRATEGY ***
330 FOR L=W TO M STEP 2
340 POKE (16384+S(L)),(3*R(L)+C(L)-3)
350 NEXT
360 POKE (16384+S(M-1)),(3*R(M)+C(M)-3)
430 IF W=Z THEN PRINT:PRINT "I WIN!    ":PRINT
440 IF W=Q THEN PRINT:PRINT "YOU WIN! ":PRINT
450 INPUT "WANT TO PLAY ANOTHER GAME";R1$
460 GOTO 310
470 ::::::::::::::::::::::::::::::::::::
480 REM *** DETERMINE WHO GOES FIRST ***
900 Z=1
910 Q=2
920 IF RND(1)<0.5 THEN 950
930 PRINT "I WILL BE X AND WILL MOVE FIRST."
940 RETURN
950 Z=2
960 Q=1
970 PRINT "YOU CAN BE X AND CAN MOVE FIRST."
980 RETURN
990 ::::::::::::::::::::::::::::::::::::
995 REM   *** SEARCH FOR A WINNER ***
1000 A=0
1010 Z3=0
1020 Z4=0
1030 FOR J=1 TO 3
1040 Z2=0
1050 Z1=0
1060 FOR K=1 TO 3
1070 IF B(K,J)=W THEN Z2=Z2+1
```

```
1080 IF B(J,K)=W THEN Z1=Z1+1
1090 IF B(K,J)=W AND J=K THEN Z3=Z3+1
1100 IF B(K,J)=W AND J+K=4 THEN Z4=Z4+1
1110 NEXT
1120 IF Z1=3 THEN A=1
1130 IF Z2=3 THEN A=1
1140 NEXT
1150 IF Z3=3 THEN A=1
1160 IF Z4=3 THEN A=1
1170 RETURN
1180 :::::::::::::::::::::::::::::::::::
1190 REM  *** COMPUTER'S MOVE ***
2000 PRINT "MY MOVE IS ";
2010 W=Z:Y=Q
2020 R2=PEEK(16384+T):IF R2=0 THEN 2040
2025 IF R2>9 THEN STOP
2030 J=INT(1+(R2-1)/3):K=R2-3*INT((R2-1)/3):R(M)=J:C(M)=K:GOTO 2110
2040 FOR J=1 TO 3
2050 FOR K=1 TO 3
2055 IF R(M)>0 THEN 2100
2060 IF B(J,K)>0 THEN 2100
2070 R(M)=J
2080 R2=3*J+K-3
2090 C(M)=K
2100 NEXT K,J
2110 B(R(M),C(M))=Z:S(M)=T:T=T+Z*3↑(3*R(M)+C(M)-4)
2120 PRINT "ROW";STR$(R(M));", COLUMN";STR$(C(M))"."
2130 RETURN
2140 :::::::::::::::::::::::::::::::::::
2150 REM  *** YOUR MOVE SUBROUTINE ***
3000 W=Q:Y=Z
3010 PRINT "YOUR MOVE. ";
3020 INPUT "WHAT ROW AND COLUMN";R(M),C(M)
3025 IF R(M)>3 OR R(M)<1 OR C(M)>3 OR C(M)<1 THEN 3060
3030 IF B(R(M),C(M))>0 THEN 3060
3040 B(R(M),C(M))=Q:S(M)=T:T=T+Q*3↑(3*R(M)+C(M)-4)
3050 RETURN
3060 PRINT "INVALID MOVE. PLEASE TRY AGAIN."
3070 GOTO 3020
3080 :::::::::::::::::::::::::::::::::::
3090 REM  *** PRINT OUT BOARD ***
4000 PRINT:FOR J=1 TO 3
4010 FOR K=1 TO 3
4020 PRINT TAB((K-1)*4+2);
4030 IF B(J,K)=1 THEN PRINT "X";
4040 IF B(J,K)=2 THEN PRINT "O";
4050 IF K<3 THEN PRINT TAB(4*K);"|";
4060 NEXT
4070 PRINT
4080 IF J=3 THEN 4100
4090 PRINT " ---+---+---"
4100 NEXT
4120 RETURN
4130 :::::::::::::::::::::::::::::::::::
```

LINE	EXPLANATION
5-45	These lines provide identification for the program.
47-320	This is the main program.
47	This is a remark to identify the section.
50	These POKEs reduce the top of the RAM to provide a section of memory to store strategy. The BASIC program will not enter these locations.
55-60	This message is printed so that you will know that the computer IS working and not waiting for you to do something.
70	G is the game number. This loop fills in the memory locations that were reserved for strategy with zeros.
80	G is increased by one. T is a variable that is used to store a code for the placement of the moves on the board.
90	This line clears the screen and prints the name of the game plus a blank line.
100	This line prints the game number.
110-170	This loop clears the board array, B, the row array, R, the column array, C, and the sum array, S.
180	M is the move number, which is set at zero.
190	These are calls to two subroutines. The first subroutine determines who moves first, and the second prints the board.
200	This line increments the move variable.
210	This line prints the move number and the two blank lines.
220	This line sets M2 as one-half the move number and M3 as either 0 or −1: 0 if M is odd, and −1 if M is even.
230	This line sets M4 to 0 if M is even, or to −1 if M is odd.
240	If it is the computer's move, control goes to the subroutine that starts at line 2000.
250	If it is your move, control goes to the subroutine that starts at line 3000.
260	Control goes to the subroutine that prints the board.
270	The subroutine called in this line checks to see if there is a winner.
280	If A equals one, there is a winner, and the computer will store the winning strategy by branching to line 330.
290	If all the moves are not taken, control goes to line 200.
300	Because all the spots have been taken and there is still no winner, the computer prints the message

	that nobody won and asks player if he wants to play again.
310	If the first letter input was Y, control goes back to line 80.
320	If the first letter was not Y, the game is ended.
325	This is a separation line made of colons.
327-460	This routine stores the winning strategy in the reserved memory locations.
327	This line identifies this section.
330-350	This loop stores the winning moves.
360	This line replaces the last move of the loser with the correct winning move. This way the computer will not ever make that move again!
430	If the computer won, the appropriate message is printed.
440	If you won, the appropriate message is printed.
450	This line asks whether or not you want to play another game.
460	Control goes to line 310 to evaluate the answer.
470	This is a separation line.
480-980	This subroutine determines who goes first by using the random number generator.
480	This line identifies this subroutine.
900	2 = 1 will make the computer move first.
910	Q = 2 will make you move second.
920	If a random fraction is less than .5, then control branches to line 950, and the order of who goes first switches.
930	This line prints that the computer will be X and will move first.
940	Control returns to the main program.
950	The computer will move second.
960	You will move first.
970	This line prints that you will be X and can move first.
980	Control is returned to the main program.
990	This is a separation line.
995-1170	This subroutine searches for a winner.
1000	A is the variable that indicates whether or not there is a winner. If A is zero, there is no winner. If A is changed to 1, there is a winner. A is a *flag* variable.
1010	Z3 is a flag that indicates a bottom left to upper right diagonal win.
1020	Z4 is used to check for an upper left to bottom right diagonal win.
1030	J is a loop for checking the columns on the board.
1040	Z2 is for checking for a vertical win.

1050	Z1 is for checking for a horizontal win.
1060	K is the loop that checks the rows on the board.
1070	If this board position has a value equal to W, Z2 is increased by one.
1080	If this board position has a value of W, Z1 is increased.
1090	If the bottom left to upper right diagonal has a value of W, Z3 is increased.
1100	If the position in the other diagonal has a value equal to W, Z4 is increased.
1110	This is the end of the K loop.
1120	This line sets A to 1 if Z1 equals 3.
1130	This line sets A equal to 1 if Z2 is 3.
1140	This is the end of the J loop.
1150	If Z3 is 3, A is set to 1.
1160	If Z4 is 3, A is set to 1.
1170	Control returned to the main program.
1180	This is a separation line.
1190-2130	This subroutine determines the computer's move.
1190	This line identifies this subroutine.
2000	This line starts to print out the computer's move.
2010	W is the computer's first move, and Y is your first move number.
2020	This line sees if there is a stored number that is a code for the move to take. T is the coded sum of the moves on the board.
2025	If the number returned from the PEEK is greater than 9, there has been an error, and the game is stopped.
2030	This line converts the R2 code back into J and K values and then makes its move based on that. It branches back to line 2110.
2040-2100	These nested loops choose the first available position on the board if the R2 value was zero.
2110	This line assigns the value to the board array and the sum array, and increases the sum variable, T, based on the X or O and the board position.
2120	This line prints the computer's move on the screen.
2130	This line returns control to the main program.
2140	This is a separation line.
2150-3070	This subroutine allows you to enter your move.
2150	This line identifies the subroutine.
3000	W is your first move, and Y is the computer's first move.
3010	This line states that it is your move.
3020	This line requests the row and column you want to move to.

3025	This line checks for an invalid row or column number.
3030	This line checks to see if the position has already been taken.
3040	This line assigns values to the board array, the sum array, and the sum variable based on the value of the move, the symbol used (X or O), and the position taken.
3050	This line returns control to the main program.
3060	This line prints the error message.
3070	This line branches back to entry line.
3080	This is a separation line.
3090-4120	This subroutine prints out the tic tac toe board.
3090	This line identifies the subroutine.
4000	J is the loop for the rows on the board.
4010	K is the loop for the columns on the board.
4020	This line establishes the next print position.
4030	If the board value is 1, an X is printed.
4040	If board value is 2, an 0 is printed.
4050	This line prints a vertical bar, which is the graphic character on the B key.
4060	This is the end of the K loop.
4070	This line prints a blank line.
4080	If the end of J loop has been reached, the horizontal line will not be printed.
4090	This line prints the graphic character found on the asterisk key and the graphic character found on the plus sign. Together these characters make the horizontal lines of the board.
4100	This is the end of the J loop.
4120	Control is returned to the main program.
4130	This is the last line of the program.

Program Operation. When the program starts, it will inform you who will be X and will move first. It will also display an empty tic tac toe board. You will be asked for the row and column of your move. Enter it as 2,3 for row 2 and column 3, for example. The rows are numbered from 1 to 3 from top to bottom, and the columns are numbered from 1 to 3 from left to right. After each of your moves the board will be updated and the computer will display its move. You play against the computer, moving alternately. After the game is over, you will be asked if you want to play again. Answer either YES or NO. Enjoy the game!

The Five-in-a-Row Game

The program in Listing 5-2 plays the game of five-in-a-row with you and up to three other human opponents. The computer

Listing 5-2 Five-in-a-Row Game

```
10 :::::::::::::::::::::::::::::::::::::
20 REM      A FIVE-IN-A-ROW GAME
30 REM WRITTEN BY TIMOTHY J. O'MALLEY
40 REM COPYRIGHT 1983, TAB BOOKS INC.
50 REM (WRITTEN FOR THE COMMODORE 64)
60 ::::::::::::::::::::::::::::::::::::
70 PRINT CHR$(147);
75 PRINT "       *** FIVE IN A ROW GAME ***"
80 PRINT:PRINT
81 PRINT "       THE GOAL OF THIS GAME IS TO PLACE"
82 PRINT "FIVE DOTS IN A ROW, EITHER VERTICALLY,"
83 PRINT "HORIZONTALLY OR DIAGONALLY. YOU MAY PUT"
84 PRINT "YOUR DOT ON ANY UNOCCUPIED POSITION."
90 PRINT "ENTER MOVES AS ROW COLUMN. FOR EXAMPLE:"
93 PRINT "H 21 WILL PLACE YOUR DOT AT ROW H AND  "
94 PRINT "COLUMN 21. IF THAT'S THE INTENDED SPOT,"
95 PRINT "PRESS THE RETURN KEY. IF NOT, PRESS ANY"
96 PRINT "OTHER KEY AND RE-ENTER. THE POSITION   "
97 PRINT "WILL FLASH THE DOT BEFORE YOU PRESS THE"
98 PRINT "RETURN KEY. THE CORNERS DISPLAY THE"
99 PRINT "CURRENT COLOR.":PRINT
100 DIM B%(22,38):REM DIMENSION ARRAY FOR BOARD
105 DIM A%(5,6),SR(5,2):REM DIMENSION ARRAYS FOR MOVES
106 DIM CR(5):FOR I=1 TO 5:READ CR(I):NEXT:REM COLORS OF DOTS
107 DATA 7, 1, 8, 13, 3
109 PRINT "THE COMPUTER WILL MOVE LAST."
110 PRINT:INPUT"WHAT NUMBER OF HUMAN PLAYERS (1-4)";NP
120 IF NP<1 OR NP>4 THEN 110
130 PRINT CHR$(147);:REM CLEAR SCREEN
135 F3=0:REM FLAG FOR START OF GAME
140 FOR I=1024 TO 2023:POKE I,81:NEXT
150 FOR I=0 TO 21:POKE(1104+40*I),(1+I):POKE(1143+40*I),(1+I)
160 POKE (55336+40*I),1:POKE(55375+40*I),1:NEXT
170 FOR I=1 TO 38
193 IF I/10>=1 THEN POKE 1024+I,INT(I/10)+48:POKE 55296+I,1
194 IF I/10>=1 THEN POKE 1944+I,INT(I/10)+48:POKE 56216+I,1
200 A=I-10*INT(I/10):A=48+A
210 POKE 1064+I,A:POKE 1984+I,A:POKE 55336+I,1:POKE 56256+I,1:NEXT
215 NC=1
220 IF NC>NP+1 THEN NC=1
230 POKE 55335,CR(NC):POKE 56295,CR(NC)
240 POKE 56256,CR(NC)
245 PRINT CHR$(19);:POKE 55296,CR(NC)
246 FOR I=1024 TO 1031:POKE I,32:NEXT
250 OPEN 1,0
260 INPUT#1,A$:PRINT
270 CLOSE 1:Z=ASC(LEFT$(A$,1))-64:IF Z<1 OR Z>22 THEN 245
280 Y=VAL(MID$(A$+" ",3,2)):IF Y<1 OR Y>38 THEN 245
290 IF B%(Z,Y)>0 THEN 245
300 POKE 1064+40*Z+Y,81:POKE 55336+40*Z+Y,CR(NC)
310 GET C$:POKE 1064+40*Z+Y,32:POKE 55336+40*Z+Y,CR(NC):IF C$="" THEN
    300
```

```
320 IF CHR$(13)=LEFT$(C$,1) THEN GOSUB 1220:GOTO 340
330 GOTO 245
333 ::::::::::::::::::::::::::::::::::::::::
335 REM  *** SEARCH FOR A WINNER ***
340 B%(Z,Y)=NC:V=0:H=0:L=0:K=0
350 FOR I=(Y-5)*-(Y>4)+1 TO (Y-34)*-(Y<35)+38
360 FOR J=(Z-5)*-(Z>4)+1 TO (Z-16)*-(Z<17)+20
370 IF B%(J,I)=NC AND I=Y THEN V=V+1:GOTO 390
375 IF V=5 THEN END
380 IF I=Y AND B%(J,I)<>NC THEN V=0
390 IF B%(J,I)=NC AND J=Z THEN H=H+1:GOTO 410
395 IF H=5 THEN END
400 IF J=Z AND B%(J,I)<>NC THEN H=0
410 IF B%(J,I)=NC AND (Y-I)=(Z-J) THEN L=L+1:GOTO 430
415 IF L=5 THEN END
420 IF (Y-I)=(Z-J) AND B%(J,I)<>NC THEN L=0
430 IF B%(J,I)=NC AND (Y-I)=(J-Z) THEN K=K+1:GOTO 450
435 IF K=5 THEN END
440 IF (Y-I)=(J-Z) AND B%(J,I)<>NC THEN K=0
450 IF H=5 OR V=5 OR L=5 OR K=5 THEN PRINT CHR$(19);:END
460 NEXT:NEXT:A%(NC,1)=Y:A%(NC,2)=Z
470 NC=NC+1:IF NC=NP+1 THEN 495:REM COMPUTER'S MOVE
480 GOTO 220
485 ::::::::::::::::::::::::::::::::::::::::
490 REM *** COMPUTER'S MOVE ***
495 POKE 55335,CR(NC):POKE 56295,CR(NC):POKE 56256,CR(NC)
496 PRINTCHR$(19);"COMPUTER";
500 IF F3=0 THEN 1100:REM START OF GAME - FIND EMPTY SQUARES
505 FY=0:FX=0
510 FOR NX=1 TO NP
520 Y=A%(NX,1):Z=A%(NX,2)
540 V=0:H=0:L=0:K=0:SM=2:FX=0:FY=0
550 FOR I=(Y-5)*-(Y>4)+1 TO (Y-34)*-(Y<35)+38
560 FOR J=(Z-5)*-(Z>4)+1 TO (Z-16)*-(Z<17)+20
570 IF B%(J,I)=NX AND I=Y THEN V=V+1:GOTO 590
580 IF I=Y AND B%(J,I)<>0 AND V>0 THEN V=0:GOTO 590
585 IF B%(J,I)=0 AND I=Y AND V>SM THEN FX=J:FY=I:SM=V
590 IF B%(J,I)=NX AND J=Z THEN H=H+1:GOTO 610
600 IF J=Z AND B%(J,I)<>0 AND H>0 THEN H=0:GOTO 610
605 IF B%(J,I)=0 AND J=Z AND H>SM THEN FX=J:FY=I:SM=H
610 IF B%(J,I)=NX AND (Y-I)=(Z-J) THEN L=L+1:GOTO 630
620 IF (Y-I)=(Z-J) AND B%(J,I)<>0 AND L>0 THEN L=0:GOTO 630
625 IF B%(J,I)=0 AND (Y-I)=(Z-J) AND L>SM THEN FX=J:FY=I:SM=L
630 IF B%(J,I)=NX AND (Y-I)=(J-Z) THEN K=K+1:GOTO 650
640 IF (Y-I)=(Z-J) AND B%(J,I)<>0 AND K>0 THEN K=0:GOTO 650
645 IF B%(J,I)=0 AND (Y-I)=(J-Z) AND K>SM THEN FX=J:FY=I:SM=K
650 NEXT:NEXT
660 A%(NX,3)=V:A%(NX,4)=H
670 A%(NX,5)=L:A%(NX,6)=K
680 NEXT
690 FOR I=1 TO NP:FOR J=3 TO 6
700 IF FX>0 AND FY>0 AND A%(I,J)=SM THEN 1200
710 NEXT:NEXT
805 FY=0:FX=0
```

```
810 FOR NX=1 TO NP
820 Y=A%(NX,1):Z=A%(NX,2)
840 V=0:H=0:L=0:K=0:SM=2:FX=0:FY=0
850 FOR I=(Y-34)*-(Y<35)+38 TO (Y-5)*-(Y>4)+1 STEP -1
860 FOR J=(Z-16)*-(Z<17)+20 TO (Z-5)*-(Z>4)+1 STEP -1
870 IF B%(J,I)=NX AND I=Y THEN V=V+1:GOTO 890
880 IF I=Y AND B%(J,I)<>0 AND V>0 THEN V=0:GOTO 890
885 IF B%(J,I)=0 AND I=Y AND V>SM THEN FX=J:FY=I:SM=V
890 IF B%(J,I)=NX AND J=Z THEN H=H+1:GOTO 910
900 IF J=Z AND B%(J,I)<>0 AND H>0 THEN H=0:GOTO 910
905 IF B%(J,I)=0 AND J=Z AND H>SM THEN FX=J:FY=I:SM=H
910 IF B%(J,I)=NX AND (Y-I)=(Z-J) THEN L=L+1:GOTO 930
920 IF (Y-I)=(Z-J) AND B%(J,I)<>0 AND L>0 THEN L=0:GOTO 930
925 IF B%(J,I)=0 AND (Y-I)=(Z-J) AND L>SM THEN FX=J:FY=I:SM=L
930 IF B%(J,I)=NX AND (Y-I)=(J-Z) THEN K=K+1:GOTO 950
940 IF (Y-I)=(J-Z) AND B%(J,I)<>0 AND K>0 THEN K=0:GOTO 950
945 IF B%(J,I)=0 AND (Y-I)=(J-Z) AND K>SM THEN FX=J:FY=I:SM=K
950 NEXT:NEXT
960 A%(NX,3)=V:A%(NX,4)=H
970 A%(NX,5)=L:A%(NX,6)=K
980 NEXT
990 FOR I=1 TO NP:FOR J=3 TO 6
1000 IF FX>0 AND FY>0 AND A%(I,J)=SM THEN 1200
1010 NEXT:NEXT
1020 GOTO 1140
1030 ::::::::::::::::::::::::::::::::::::::
1100 Z1=7+INT(8*RND(1)):Y1=7+INT(24*RND(1)):F3=1
1110 Z2=INT(3*RND(1))-1:Y2=INT(3*RND(1))-1
1115 IF Z2=0 AND Y2=0 AND CT>1 THEN 1140
1117 IF Z2=0 AND Y2=0 THEN 1100
1120 FOR I=1 TO 5:Z3=Z1+Z2*(I-1):Y3=Y1+Y2*(I-1)
1125 IF B%(Z3,Y3)>0 THEN IF B%(Z3,Y3)<>NC THEN 1110
1130 SR(I,1)=Z3:SR(I,2)=Y3:NEXT:CT=1
1140 IF B%(SR(CT,1),SR(CT,2))=0 THEN B%(SR(CT,1),SR(CT,2))=NC:CT=CT+1:
     GOTO 1190
1150 CT=CT+(CT>1):Z1=SR(CT,1):Y1=SR(CT,2):GOTO 1110
1160 ::::::::::::::::::::::::::::::::::::::
1190 Z=SR(CT-1,1):Y=SR(CT-1,2):GOSUB 1220:GOTO 340
1200 Z=FX:Y=FY:GOSUB 1220:GOTO 340
1210 ::::::::::::::::::::::::::::::::::::::
1220 POKE 1064+40*Z+Y,81:POKE 55336+40*Z+Y,CR(NC):RETURN
1230 ::::::::::::::::::::::::::::::::::::::
```

will act as another opponent. Each player will try to place five dots in a row on the board to win. Each player will be assigned a specific color based on his or her move number. The color of the current player will be displayed in the corners during his or her turn. The players take turns, with the computer moving last. The more players there are, the longer it takes the computer to decide where to place its dot. If a player has three or more dots in a five dot space, the computer will attempt to block his or her next move. The program will check for a winner after each player has made a move.

When it is your turn, select the row and column of your proposed move by typing something like J 12. Then press the RETURN key. If the letter or number is invalid, the computer will keep on blinking the square in the upper left corner and seek another input. A dot will appear and blink rapidly in the appropriate position after valid a valid row and column numbers are entered. If you want to move there, press RETURN again and the dot will remain. Otherwise press another key to reselect. The first player getting five dots in a row wins the game, and the program will stop.

LINE	**EXPLANATION**
10-60	This is identification information for the program.
70	This line clears the screen.
75	This line prints the name of the game on the screen.
80	This line prints two blank lines.
81-99	These lines print the instructions for the game.
100	This line dimensions the array to store the moves on the board. The board uses 22 rows and 38 columns.
105	This line dimensions the arrays used in storing moves and in keeping track of the computer's anticipated moves.
106	This line dimensions an array to store the colors of the dots for each player. The I loop reads in the color codes for up to five players, including the computer.
107	This DATA statement contains the color code data for the CR array.
109	This line tells you that the computer will move last.
110	This line requests the number of human players, NP.
120	This is an error routine in case you entered the wrong number of players (outside the range of 1-4).
130	This line clears the screen.
135	F3 is a special flag used for the start of the game.
140	This line fills the entire screen with dots that are not turned on.
150-160	This I loop prints the letters that label the 22 rows.
170-210	This I loop fills in the numbers for the 38 columns on the screen.
215	NC is the number of the player whose turn it now is.
220	If the computer has just played, the player-number counter is reset.
230-240	These lines color the three corners (top right, bottom

right, and bottom left) with the color of the current player.

245	This line prints the home character and colors the upper left corner with the color of the current player.
250-260	These lines read the keyboard for the move position.
270	This line closes the keyboard file and finds the value of Z using the letter that was entered for the row. Z is now the index of the row for the move. If an invalid letter or character was entered, the keyboard is reread.
280	Y is the number of the column that was entered. If an improper number was entered, the computer will reread the keyboard.
290	If the position just entered is already occupied, control goes back to line 245, and the keyboard is reread.
300	This line places a colored dot at the indicated position.
310	This line reads the keyboard, and blanks the dot out. If a key is not pressed, control goes to line 300. This will cause the dot to blink on and off rapidly until a key is pressed.
320	If the key pressed was the RETURN key, control goes to the subroutine at 1220 and then to line 340.
330	If it was another key, control goes to line 245 to read the keyboard again.
333	This is a separation line.
335-480	This subroutine searches for a winner after each person makes his move.
335	This line identifies this subroutine.
340	This line stores the player's position in the board array and sets the flag variables for the vertical, horizontal, and diagonal indicators to zero. If one of these variables becomes 5, there is a winner.
350	The I loop searches up to 5 columns to the left and to the right for determining the winner.
360	The J loop searches up to 5 rows above and below.
370	If there is a player's dot in the same column as this move occupies, then add 1 to V and branch to 390.
375	If V reaches 5, there is a winner and the game is over.
380	If the dot in that column is someone else's, V is reduced to zero because of a "block."
390	If one of the player's own dots is in the same row, it is increased by 1 and control goes to 410.
400	If H is 5, the game is ended.

410	If a dot is in the game diagonal line, L is increased by 1 and control goes to 430.
415	If L is 5, the game is ended.
420	If that dot belonged to another player L is reduced to zero.
430	If a dot is in the opposite diagonal, K is increased by 1, and control branches to 450.
435	If K is 5, the game is ended.
440	If that dot is someone else's, K is reduced to zero.
450	If any of the variables equals 5, the home character is printed and the game is ended.
460	This line ends both loops and sets two elements of the A% array to the row and column positions of this player.
470	This line increases the player number by one. If it is the computer's turn, control branches to line 495.
480	Otherwise control goes to line 220.
485	This is a separation line.
490-1020	This subroutine determines the computer's move.
490	This is the name of the subroutine.
495	This line colors the corners the computer's color.
496	This line prints the word COMPUTER in the upper left corner.
500	If it is the start of the game, control goes to the subroutine at line 1100 to find 5 blank squares for the computer's proposed moves.
505	This line set the computer's row and column choice to zero.
510	NX is a loop to check out all of the other player's moves.
520	Y and Z are set as the row and column of the player's last move, as stored in array, A%.
540	This line sets various variables to zero. SM is the sum variable, which is used to check the amount of dots that a player has in a row.
550	this I loop checks the columns up to five positions from the player's last move.
560	This J loop checks the rows up to five squares above or below the player's last move.
570-645	These lines are like the lines used to check for a winner in the routine in lines 335-480. FX and FY are set to the row and column numbers of blank positions that can be used to block the opponent's moves. The computer will attempt to block if a player has 3 or more dots within a space of 5 squares in a row, column, or diagonal.
650	This line then ends the J and I loops.

660-670	These lines store the number of dots for the vertical, V, the horizontal, H, and the two diagonals of each player in the A% array.
680	This is the end of the NX loop.
690-710	These nested loops check to see where a possible move to block an opponent was recorded in A%. If one was found, a branch to line 1200 is made.
805-1010	These lines are run if the computer did not find a counter move. These lines examine the series of moves from the other direction, because this possibility was not taken into account by the first routine. Notice that the I and J loop have STEP — 1 and go from the end to the beginning, relative to the first set.
1020	If a counter move was not found or was not appropriate, the computer branches to line 1140 to make its own line of moves.
1030	This is a separation line.
1100-1150	These lines select and fill five unoccupied positions on the board for the computer.
1100	This line selects a starting row and column position for the computer. It sets the start flag variable. F3, to 1 to indicate that it is no longer the start of the game.
1110	Z2 and YY are each an integer, −1, 0,or 1. These represent the increment added to the row and column from the starting row and column position.
1115	If Z2 and Y2 are both zero, and the computer has already filled at least one of its positions, control branches to line 1140.
1117	Otherwise, a new series of squares is obtained for the moves.
1120	This I loop positions the dot or checks for an opponent's dot in one of the computer's proposed positions. Z3 and Y3 are the new row and column of the proposed move.
1125	If that position is taken by another's dot, control branches to line 1110.
1130	This line stores the move in array SR and sets CT to 1.
1140	This line stores the proposed move in the board array, B%, increases CT by one, and branches to line 1190.
1150	This line subtracts 1 from CT, if CT is greater than 1, and then starts a line of dots from that position.
1160	This is a separation line.
1190-1200	Line 1190 sets the z and Y variable at the last stored SR position, calls the subroutine at line 1220, and then branches to line 340. Line 1200 does a similar

	thing if FX and FY were used above.
1210	This is a separation line.
1220	This line prints a dot on the board and colors it with the player's color. Control then returns to the calling section of the program.
1230	This is the last line of the program.

Program Operation. After you enter the program and type RUN, the computer will display the name of the game and the instructions. It will request the number of human players and then proceed with the game. Players must try to develop strategy to win the game. The computer will not save strategy in this game but will attempt to win by blocking opponents and making its own line of five dots to win. Players must try to stop each other and the computer.

Chapter 6

Pattern Recognition

In this chapter you will find two short programs concerning patterns in numbers and in literal strings, and a long program that you might find useful for word processing or text editing. In that program you can search for a particular word in the text and have the computer change all occurrences of that word.

WAYS OF FINDING PATTERNS

Pattern recognition is an area of artificial intelligence. If someone gave you a series of numbers, could you predict the next number in that series? Chances are that you could. You might notice a pattern in the sequence and then extrapolate the next number. When dealing with words, you might know what the person is going to say next, or you might recognize patterns that help you understand what the person is talking about.

Differences Between Numbers

The first program that we will examine will find patterns in numbers by looking at the differences in value between the numbers in the series. Then it will look at the differences between successive differences and so forth until there is one value left. Then the computer will use that value and "work forward" again to predict the next value. It then asks you if that was the value that you were looking for. If it is not, it will search through some other stored sequences to attempt to answer the problem. Finally it will ask you for the value. Finding differences between numbers is one way of finding patterns. You might look for multiplicative differences between numbers as well.

The Grouping of Strings

In the Perfect Logic program, you entered names of things and the computer grouped things in sets based on what you entered. This is a good example of looking for patterns in strings. The computer broke up your sentences or questions and attempted to answer or respond to your input. Language processing, then, overlaps the area of pattern recognition.

There are other ways to find patterns in strings. One way is to store strings along with whatever follows them. The next time that the computer recognizes the string, it will pring out whatever followed it. In the second program in this chapter, you enter the letters from A to G, which represents the musical notes. After you enter the same set of letters again, the computer will predict the next letter that you will type. This is a simple example of string recognition.

The final program is a word processor/text editor. In that program you can easily replace incorrectly spelled words or phrases in whatever lines that you like, and the computer will change those lines of text. This program is a bit out of the mainstream of artificial intelligence, but it is an interesting and useful area to learn about. You might consider an artificially intelligent word processor, one that would outthink the one in this book.

Interpolation and Extrapolation

Many times a value that you want to find can be found by *interpolation* or by *extrapolation*. Interpolation is finding an answer between two similar or close answers. For example if you wanted to predict a number between two numbers, the best guess that you could give would be the average or mean of those two numbers. If you want to find a value outside of two values, you might construct a regression line and use that mathematical formula to calculate the value in question. All of this leads into the area of statistics.

Statistics is a very powerful tool for predicting where numbers will fall and how events will occur. It can be used in conjunction with artificial intelligence concepts—there is simply so much territory to cover that the subject is out of the scope of this book.

PATTERN RECOGNITION PROGRAMS IN BASIC

Let's examine the three pattern recognition programs written in BASIC for the Commodore 64. The programs deal with patterns in numbers, patterns in strings, and patterns as they are used in a word processor.

Patterns in Numbers

The program in Listing 6-1 allows you to enter five numbers of a series. The computer will try to predict the sixth number in the series. If it can't find that number, it requests an answer from you.

This program displays artificial intelligence by calculating or retrieving this sixth number. The program stores answers that are different from the one that it calculates. That way, the next time you enter that series of numbers, the computer can retrieve the answer. The same series can have different sixth numbers. The computer will search down through the list and quiz the user on possible answers.

Here's a line by line explanation of the program.

Listing 6-1 Patterns in Numbers

```
10 :::::::::::::::::::::::::::::::::::::
20 REM  *** PATTERNS IN NUMBERS ***
30 REM WRITTEN BY TIMOTHY J. O'MALLEY
40 REM COPYRIGHT 1984, TAB BOOKS INC.
50 REM (WRITTEN FOR THE COMMODORE 64)
60 :::::::::::::::::::::::::::::::::::::
70 DIM A(6,6),B$(100),C$(100):CT=0
75 PRINT "(CLR)";TAB(9)"*** NUMBER GUESSER ***":PRINT
80 PRINT "      ENTER FIVE NUMBERS, ONE AT A TIME."
90 PRINT "I WILL TRY TO PREDICT THE SIXTH NUMBER"
100 PRINT "IN THE SERIES.":PRINT
110 FOR J=1 TO 5:INPUT A(1,J):NEXT
120 FOR I=2 TO 5:FOR J=1 TO 6-I
130 A(I,J)=A(I-1,J+1)-A(I-1,J)
140 NEXT J,I
150 A(6,1)=A(5,1)
160 FOR I=5 TO 1 STEP -1
170 A(I,7-I)=A(I,6-I)+A(I+1,6-I)
180 NEXT
190 H=A(1,6):PRINT:PRINT "IS IT"STR$(H);
200 INPUT A$
210 IF LEFT$(A$+"Y",1)="Y" THEN PRINT "GOOD!":PRINT:GOSUB 340:GOTO 75
220 N$="":FOR N=1 TO 5:N$=N$+STR$(A(1,N)):NEXT
230 IF CT=0 THEN 300
240 FOR I=1 TO CT
250 Y$=B$(I):IF N$<>Y$ THEN 290
260 X$=C$(I):PRINT "IS IT "X$;
270 INPUT A$
280 IF LEFT$(A$+"Y",1)="Y" THEN PRINT "GREAT!":PRINT:GOSUB 340:GOTO 75
290 NEXT
300 INPUT "WHAT IS THE ANSWER";Z$
310 CT=CT+1:B$(CT)=N$:C$(CT)=Z$
320 PRINT "I'LL REMEMBER THAT!":PRINT:GOSUB 340:GOTO 75
```

```
330 ::::::::::::::::::::::::::::::::::::
340 FOR I=1 TO 5000:NEXT:RETURN
350 ::::::::::::::::::::::::::::::::::::
```

LINE	EXPLANATION
10-60	This is the program identifying information.
70	This line dimensions the arrays used in the program. A is an array used to store the 5 numbers that you enter. B$ is an array for storing up to 100 special series of numbers. C$ is the array that stores the sixth number in each of the stored series in B$. CT is a variable used as a counter for the number of stored series in B$.
75	This line prints the CLR character and prints the title of the program on the screen.
80-100	These lines print the instructions.
110	This line lets you enter the 5 numbers into the A array.
120-140	These lines find the arithmetic difference between the numbers in the series, then find the differences between those differences, and so forth until it gets down to one number.
150	The last number is duplicated in the sixth row of the array.
160-180	The process is reversed with the duplicated number being added to the differences to calculate the sixth number in the series.
190	This line asks if its number is correct. STR$ was used so that there would be no space between the number and the question mark of the input.
200	This line solicits a YES or NO answer to the above question.
210	If the first letter of your reply is Y, the computer prints GOOD! then calls the subroutine at line 340 and branches back to line 75 for another series of numbers.
220	This line stores the series of numbers as N$.
230	If no series of numbers were previously stored, the computer branches to line 300 to request the answer.
240-290	This I loop runs through the stored series of numbers searching for a match. When it finds one, it prints out the answer stored in C$. If that is the answer, the computer replies, GREAT! then calls line 340 and goes to line 75.
300	If the answer is not found, the computer gives up and asks you for the answer.

310	This line increments the CT counter and stores the series in B$ and the answer in C$.
320	The computer acknowledges the input of the correct number, calls line 340, and branches to line 75.
330	This is a separation line.
340	This subroutine provides a time delay of a few seconds before continuing the program. You might use TI$ as a timer instead. You would set TI$ and then check it against a predetermined value.
350	This is the last line of the program.

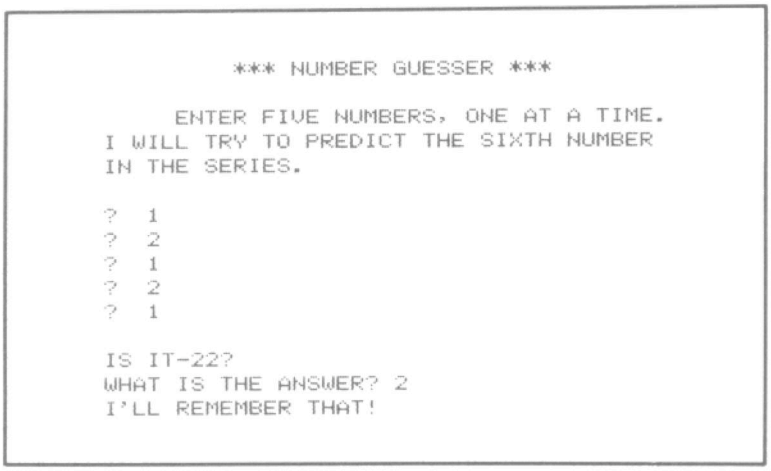
Program Operation. When the program is entered and RUN, the computer will display the program title and the instructions. You are to enter five numbers, one at a time into the computer. The computer will ask you if a certain answer is correct.

Figures 6-1 and 6-2 show the operation of the program. Figure 6-1 shows the first time the series was entered, and Fig. 6-2 shows the second time the series was entered. You might want to change the program to search through the stored answers before trying to calculate an answer. You might work backwards in the list to retrieve the last answer stored; use a loop with a STEP −1.

Patterns in Strings

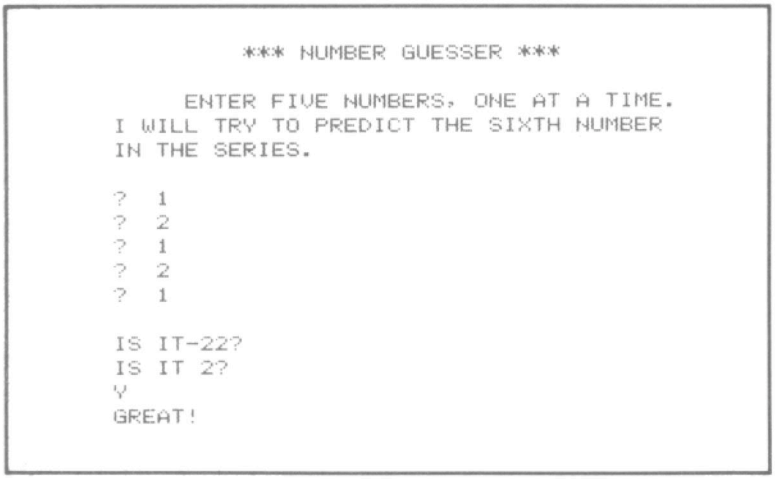
The short program shown in Listing 6-2 attempts to determine the next letter that you will press based on what you pressed before.

The computer will take the last four letters that you entered and convert them to a code to store in memory. If you enter four letters that have been saved before, the computer will print out (in cyan) the letter that it thinks you will type next.



```
              *** NUMBER GUESSER ***

         ENTER FIVE NUMBERS, ONE AT A TIME.
     I WILL TRY TO PREDICT THE SIXTH NUMBER
     IN THE SERIES.

     ?  1
     ?  2
     ?  1
     ?  2
     ?  1

     IS IT-22?
     WHAT IS THE ANSWER? 2
     I'LL REMEMBER THAT!
```

Fig. 6-1. The first time that a series is entered into Listing 6-1.



```
           *** NUMBER GUESSER ***

       ENTER FIVE NUMBERS, ONE AT A TIME.
    I WILL TRY TO PREDICT THE SIXTH NUMBER
    IN THE SERIES.

    ?  1
    ?  2
    ?  1
    ?  2
    ?  1

    IS IT-22?
    IS IT 2?
    Y
    GREAT!
```

Fig. 6-2. The second time a series of numbers is entered into Listing 6-1.

Listing 6-2 Patterns in Strings

```
5 ::::::::::::::::::::::::::::::::::::::
10 REM  *** PATTERNS IN STRINGS ***
20 REM WRITTEN BY TIMOTHY J. O'MALLEY
30 REM COPYRIGHT 1984, TAB BOOKS INC.
40 REM (WRITTEN FOR THE COMMODORE 64)
50 ::::::::::::::::::::::::::::::::::::::
55 DIM Z%(16807):C=0
60 LL=0:PRINT "{CLR}{WHT}ENTER LETTERS A-G.{CYAN}"
70 GET A$:IF A$="" THEN 70
80 A=ASC(A$)-64:IF A<0 OR A>7 THEN 70
90 C=C+1:PRINT A$;
100 M$=RIGHT$(" 0 0 0 0"+M$+STR$(A),10)
110 NL=0:FOR I=0 TO 4
120 NL=VAL(MID$(M$,I*2+1,2))↑I+NL
130 NEXT:IF Z%(NL) THEN PRINT "{WHT}"CHR$(Z%(NL)+64)"{CYAN}"
140 Z%(LL)=A:LL=NL:GOTO70
150 ::::::::::::::::::::::::::::::::::::
```

This program might be useful when you are experimenting in music because the letters entered must be in the range of letters from A to G.

Here is a line by line explanation of the program.

LINE	EXPLANATION
5	This is a separation line.
10-50	This is program identification information.
50	Z% is a very large array used to store all the possible combinations of the codes of the letters A-G.
55	This is a separation line.

60-70	These lines prompt you to enter letters from A to G.
80	If the letters are not within this range, they are rejected and the program branches back to line 70.
90	This line prints out the character that you entered.
100	M$ is the string array of the four letters that you entered. If you just started entering letters, M$ is padded with zeros.
110-130	The I loop converts the numbers to a single value, NL. If the value of Z% at index NL is found to be greater than 0, the computer converts it to a letter and prints it out in white.
140	The current four letters are stored in Z%, LL is set to NL, and the program branches to line 70.
150	This is the last line of the program.

Program Operation. After you enter the program and type RUN, the computer will prompt you to enter some letters from A to G. You can type these; if a four letter group matches a group stored in memory, the next letter from that group will be printed in cyan. The ones that you entered are printed in white.

A Word Processor Program

The program shown in Listing 6-3 works as a word processor/text editor. It makes your Commodore 64 become a simple, yet powerful tool. You can enter text, print the text on a printer, view the current text on the screen, save the text on cassette tape, insert lines of text, load text from cassette tape, delete a line of text, edit a line of text, change words in the text, or move lines of text. Listing 6-4 shows the lines that you will have to change or insert to use the program with a disk drive. All the rest of the program would remain the same.

Listing 6-3 A Word Processor

```
10 ::::::::::::::::::::::::::::::::::::
20 rem   word processor version 7
30 rem written by timothy j. o'malley
40 rem copyright 1984, tab books, inc.
50 rem (written for the commodore 64)
60 ::::::::::::::::::::::::::::::::::::
70 dim c$(550):c=1:printchr$(14);chr$(5);chr$(8);
80 bf=fre(0):bf=bf-(bf<0)*65536:printchr$(147);tab(5);"***";bf;"BYTES
   FREE ***"
85 if bf<500 then print:print" ***  LOW MEMORY! SAVE TEXT!  ***":goto
   90
86 print
90 print:print"     WORD PROCESSOR - VERSION 7 "
91 print:print"     f1 - PRINT text on printer":print
92 print"     f2 - VIEW current text":print
```

```
93 print"      f3 - SAVE text on tape":print
94 print"      f4 - INSERT a line of text":print
95 print"      f5 - LOAD text from tape":print
96 print"      f6 - DELETE a line of text":print
97 print"      f7 - EDIT a line of text":print
98 print"      f8 - MOVE lines of text"
99 print:print"      (or simply TYPE text now)":print
100 b$="":print"_";
110 get a$:if a$="" then 110
115 lb=len(b$)
120 ifa$=chr$(20)thenb$=left$(b$,lb+(lb>0)):c$(c)=b$:printa$;a$;"_";:
    goto110
125 if lb>0 then 145
130 if a$=chr$(133) then 164:rem f1
135 if a$=chr$(134) then 200:rem f3
136 if a$=chr$(135) then 280:rem f5
137 if a$=chr$(136) then 370:rem f7
138 if a$=chr$(137) then 520:rem f2
139 if a$=chr$(138) then 650:rem f4
140 if a$=chr$(139) then 790:rem f6
141 if a$=chr$(140) then 890:rem f8
145 ifa$=chr$(13)thenprintchr$(20);a$:c$(c)=c$(c)+" "+a$:c=c+1:goto100
150 ifa$=" "andlb>65thenprintchr$(20):c$(c)=c$(c)+" "+chr$(13):c=c+1:
    goto100
160 printchr$(20);a$;"_";:b$=b$+a$:c$(c)=b$:goto 110
164 print"{CLR}";
165 print"  *** TEXT PRINTING ROUTINE ***":print
166 input"Press RETURN when ready, OK";f$
170 open4,4,7:cmd4
180 fori=1toc:printc$(i)
185 ifi/30=int(i/30)thenforj=1to6:print:nextj
187 nexti
190 print#4:close4:goto 80
200 print"{CLR}";
210 print"   *** TAPE FILE SAVING ROUTINE ***":print
220 d$="file":input"What file name";d$
230 open1,1,1,d$
240 for i=1 to c
245 z$=c$(i)
250 print#1,left$(z$,len(z$)+(len(z$)>0))
260 next i
270 close1:goto 80
280 print"{CLR}";
290 print"   *** TAPE FILE LOADING ROUTINE ***":print
300 d$="":input"What file name";d$
310 open1,1,0,d$
320 get#1,e$
330 c$(c)=c$(c)+e$
340 ife$=chr$(13)thenc=c+1
350 if st=0 then 320
360 c=c-1:close1:goto 80
370 print"{CLR}";
380 print"   *** LINE EDITING ROUTINE ***":print
385 gosub 390:goto450
```

```
390 print"Current text contains";c-1;" lines.":print:f$="n"
395 input"Want display of text (y/n)";f$:print
396 if f$="n" then return
400 l1=1:input"Start at line";l1:print
410 l2=l1:input"End at line";l2
415 print"(CLR)";
420 fori=l1 to l2
430 print i;") ";c$(i);
435 if len(c$(i))=0thenprint
440 next i:print:return
450 f$="n":input"Want to change an entire line (y/n)";f$
460 iff$="n"then1090
470 l3=l1:input"Which line";l3
480 print:print"Simply type in the new line no.";l3;"now."
482 gosub 485:goto500
485 b$="":print"_";
490 geta$:ifa$=""then490
491 lb=len(b$)
492 ifa$=chr$(20)thenb$=left$(b$,lb+(lb>0)):c$(l3)=b$:printa$;a$;"_";:
    goto 490
494 ifa$=chr$(13)thenprintchr$(20);a$:c$(l3)=c$(l3)+" "+a$:return
496 printchr$(20);a$;"_";:b$=b$+a$:c$(l3)=b$:goto490
500 print:print"Line no.";l3;"is now changed.":print
510 goto370
520 print"(CLR)";
530 print"   *** TEXT VIEWING ROUTINE ***":print
540 print"Current text contains";c-1;"lines.":print
545 ld$="y":input"Want line numbers displayed (y/n)";ld$:print
550 print"Press any key to display up to 10 lines"
560 geta$:ifa$=""then560
570 fori=1toc-1 step 10
580 printchr$(147);
590 forj=i to i+9
595 ifld$="y"thenprintj;") ";
600 printc$(j);
605 if c$(j)="" then print
610 nextj
615 print:print"Press any key to continue."
620 geta$:ifa$=""then620
630 nexti
640 goto80
650 print"(CLR)";
660 print"   *** LINE INSERTION ROUTINE ***":print
665 gosub 390:print:f$="n"
670 print"Do you wish to insert a line of"
680 input"text within the current text (y/n)";f$
690 if f$="n"then 80
700 nb=0:input"Before which line number";nb
710 if nb<1 or nb>550 then 700
720 print:print"Simply type in the new line."
730 l3=0:gosub485
735 print:print"Line is being inserted."
740 fori=c to nb step -1
```

```
750 c$(i)=c$(i-1)
760 nexti
770 c$(nb)=c$(0)
780 c$(0)="":c=c+1:goto650
790 print"(CLR)";
800 print"    *** LINE  DELETION  ROUTINE ***":print
810 gosub 390:print:f$="n"
820 input"Do you wish to delete a line (y/n)";f$
830 if f$="n" then 80
840 nb=0:input"Which line number";nb
850 if nb<0 or nb>550 then 840
855 print:print"Line no.";nb;"is being deleted."
860 for i=nb to c-1
870 c$(i)=c$(i+1)
880 nexti:c=c-1:goto790
890 print"(CLR)";
900 print"    *** LINE  MOVING  ROUTINE ***":print
910 gosub 390:print:f$="n"
920 input"Do you want to move some lines (y/n)";f$
930 if f$="n" then 80
940 s1=0:input"Starting at what line no.";s1
945 ifs1<0 or s1>550 then 940
950 s2=0:input"Ending at what line no.";s2
955 if s2<0 or s2>550 or s2<s1 then 950
960 input"Inserted before what line no.";nb
965 if nb>=s1 and nb<=s2 then 960
966 if nb<1 or nb>550 then 960
970 print:print"Lines are now being moved."
980 t1=-1*(nb<s1):t2=-1*(nb>s1)
990 for i=s1*t1+s2*t2 to s2*t1+s1*t2 step t1-t2
1000 c$(0)=c$(i)
1010 for j=i+t2-t1 to nb-t2 step t2-t1
1020 c$(j+t1-t2)=c$(j)
1030 nextj
1040 c$(nb-t2)=c$(0)
1050 nb=nb+t1-t2
1060 next i
1070 c$(0)=""
1080 goto890
1090 print:input"Want to change a word/phrase (y/n)";f$:print
1100 iff$="n"then80
1110 input"Start at line";l1:print
1120 l2=l1:input"End at line";l2:print
1130 print"What ORIGINAL word or phrase?":gosub1220:print:f1$=" "+f1$+
     " "
1140 print"What NEW word or phrase?":gosub1260:print:f2$=" "+f2$+" "
1150 lf1=len(f1$):printchr$(147);:fori=l1 to l2
1160 l1$=" "+c$(i)+" ":ll1=len(l1$)
1170 if ll1<lf1then1210
1180 for j=1toll1-lf1
1190 iff1$=mid$(l1$,j,lf1)then1300
1200 next j:printc$(i);
1210 next i:goto 80
1220 f1$=""
```

```
1230 get a1$:if a1$="" then 1230
1240 if a1$=chr$(13) then return
1250 printa1$;:f1$=f1$+a1$:goto 1230
1260 f2$=""
1270 get a2$:if a2$="" then 1270
1280 if a2$=chr$(13) then return
1290 printa2$;:f2$=f2$+a2$:goto 1270
1300 l1$=left$(l1$,j-1)+f2$+right$(l1$,l11-lf1+1-j)
1310 c$(i)=mid$(l1$,2,len(l1$)-2):printc$(i);:goto 1210
```

Listing 6-4 Changes To Use the Program on Disk

```
93 print"     f3 - SAVE text on disk":print
95 print"     f5 - LOAD text from disk":print

210 print"    *** DISK FILE SAVING ROUTINE ***":print
211 if df=0 then 219
212 print "Do you want to save this file under the same filename that
    you "
213 input "just used";sn$
214 ifleft$(sn$,1)="y" then d$=df$:open15,8,15:print#15,"s0:"+d$:
    close15:goto230
219 print"▮Do not use the name of a file that is already on the disk.
    ▮"
220 d$="":input"What file name";d$
221 if d$="" then 220
222 df$=d$:df=1
230 open2,8,2,"0:"+d$+",s,w"
240 for i=1 to c
245 z$=c$(i)
250 print#2,left$(z$,len(z$)+(len(z$)>0))
260 next i
270 close2:goto 80
280 print"♥";
290 print"    *** DISK FILE LOADING ROUTINE ***":print
300 d$="":input"What file name";d$
305 df$=d$:df=1
310 open2,8,2,"0:"+d$+",s,r"
320 get#2,e$
330 c$(c)=c$(c)+e$
340 ife$=chr$(13)thenc=c+1
350 if st=0 then 320
360 c=c-1:close2:goto 80
```

This program will print upper- and lowercase letters on the screen. You simply start typing what you want. The program is constructed to output up to 80 characters per line of text. You may change the program to alter this. Two screen lines of text constitute one line of printed text. As you enter the text, the computer will automatically break the lines at the first space after the 65th character printed on the current line of text. That means that you simply have to keep typing without worrying about margins.

When the text is printed out, there will be 30 lines of text per page (8.5x11) with a break between sheets.

You can load new text from cassette tape (or disk if you use the lines in Listing 6-4) at the end of the current text. You may then move those lines of text to wherever you want. The program is designed to hold up to 550 lines of text before the Commodore runs out of memory space. There is a warning when the memory is low. You can then save the text or print it out.

When you are typing the program in for the first time, you should press the Commodore key (C=) at the lower left of the keyboard along with the SHIFT key to switch the computer into upper/lowercase character mode. Then enter the program as shown in Listing 6-3. A description of the lines of the program follows:

LINE	**EXPLANATION**
10-60	These lines identify the program.
70	This line dimensions the string array, C$. C$ will hold up to 550 lines of text. Each line in this program has up to 80 characters. When it prints out, there will be 30 lines on each 8.5 by 11 inch sheet of paper. The variable C is a counter for numbering the lines of text in the C$ array. CHR$(14) is the command to switch to lower case. CHR$(5) is the command to print in white. (You might want to change this code to something else.) CHR$(8) disables the use of the Commodore & SHIFT key combination. This keeps everything in the upper/lowercase character mode.
80	BF is the number of bytes free. This line prints the number of bytes of memory left for use in the text at the top of the "menu" screen.
85	If the available bytes left fall below 500, a special message is displayed telling you to save the text. Then you can press the RUN/STOP button, type RUN, and press RETURN to start new text. Otherwise you will run out of memory space and your work could be lost. You might want to modify this program to automatically save the text on tape or disk when the memory is low.
86	This line prints a blank line.
90-99	These lines print the name of the program and the different modes of operation. By pressing one of the f (function) keys, you can alter the operation of the program. When you are typing text, press one of the f keys immediately after pressing the RETURN key. (The RETURN key ends a paragraph or skips a

	line.) You may also simply start typing. What you enter will be at the bottom of what you entered beforehand.
100	B$ is an empty string; this line prints the underline character to act as a cursor.
110	This line seeks to read a character from the keyboard. If no key is pressed, the program keeps running this line.
115	LB is the length of B$.
120	If A$ is the DELete character (code 20), the last character is dropped from B$, the current line of text is made the same as B$, the delete character is printed twice, the underline character (the graphic character under the @ sign) is printed, and control goes to line 110.
125	If the length of B$ is greater than zero, control goes to line 145 (in other words, after a RETURN and a pressed key).
130-141	These lines cause the computer to branch to various routines based upon the function key that was pressed. f1 is for printing the text on a printer; f2 is for viewing the current text on the screen; f3 is for saving the current text on cassette tape; f4 is for inserting a line of text within the current text; f5 is for loading some text from cassette tape; f6 is for deleting a line of text from the current text; f7 is for editing the text; f8 is for moving lines of text within the current text.
145	If you pressed the RETURN key, the computer will erase the cursor from the last line, print the RETURN character, store the line of text in C$, increment the line counter and branch to line 100.
150	If you pressed the space bar and the length of the current line was over 65 characters long, it's time to go to a new line of text. The computer will erase the cursor and add a RETURN character at the end of that line of text. It will increment the counter and branch to line 100.
160	If the key that you pressed while typing was the DEL key, the program will erase the last character, print the cursor at the old position, update C$ and B$, and branch to line 100. This is the end of the main program.
164-190	This subroutine prints out the current text on paper using a Commodore compatible printer. (I used an EPSON printer and a CARDCO interface, but many

others will undoubtedly work without any trouble at all.)

164	This line clears the screen.
165-166	These lines identify the mode and give you time to turn on the printer and make sure that it is ready.
170	This line opens a file to the printer and instructs the printer to print in upper and lowercase characters. CMD4 means that all the following commands are for the printer.
180-187	These lines print out all of the lines of the stored text. After every group of 30 lines, six blank lines are printed. This way there are 30 double spaced lines per page with margins at the top and bottom of each sheet. You can change this if you like.
190	This line closes the printer file properly and branches back to the main program.
	200-270
200-270	This routine saves the current text as a tape file. You might want to convert this if you want to save your text on disk.
200	This line clears the screen.
210-220	These lines identify the mode and request the name of tape file. If you simply press RETURN, the file will be saved under the name, "file".
230	This line opens the file for saving on cassette tape.
240-260	These lines save the text on tape.
270	This line properly closes the tape file and is the end of the tape saving routine.
280-360	This routine loads tape files into the memory of the computer.
280	This clears the screen.
290-300	These lines print the name of this routine and requests the name of the tape file to be loaded. If nothing is entered, then the computer will load the first file that it comes to on the tape.
310	This command opens the tape read file.
320	The computer will read one character at a time from the tape using the GET# command.
330	The character that it gets is appended to the end of the current string member.
340	If that character is RETURN, then it marks the end of that line of text and C is incremented by one.
350	If the status of the tape is 0, indicated by the Commodore variable ST, the computer continues reading the tape file by branching to line 320.
360	Because the end of the file was read, C is decreased

	by one, the file is closed properly, and control branches to line 80.
370-510	This is the line editing routine.
370	This clears the screen.
380	This line prints the name of the routine.
385	This line calls the subroutine that starts at line 390 and then branches to line 450.
390-440	This subroutine displays the number of current lines of text and is used in some of the other routines. You specify what lines you want to display. The line number, a right parenthesis and the line of text will be printed on the screen.
450-460	These two lines ask you if you want to change an entire line of text. You reply either y or n.
470	You are asked which line to replace. Notice the use of the value of L1 as a default value if you simply press the RETURN key.
480-482	You enter the new line, and then the program branches to line 500.
485-496	This routine allows you to enter the new line.
500	This line acknowledges that the line has been changed.
510	Control branches back to the beginning of the line editing routine.
520-640	This routine allows you to view the entire text on the screen, 10 lines at a time. It's fairly straight forward and won't need a thorough explanation. You may request to have the line numbers displayed or not displayed. You press any key to display the next screen of text.
650-780	This routine allows you to insert a line of text within the current text. You can have some of the lines displayed. To insert a line of text, you specify the number of the line before which you want to insert the line. The line will then be inserted before that line. All of the other lines will be renumbered.
790-880	This routine allows you to delete a line of text. You specify the line number and the computer will delete it and move the lines that follow it to form a continuous text. The program tells you that the line is being deleted.
890-1080	This routine allows you to move some of the lines of text to another place in the text. You specify the starting and ending line numbers of the lines to be moved and the line number where they are to be inserted. Notice the use of certain logical conditions

(line 980) to tell the computer where the lines that have to be moved are located. This is necessary to have the text become continuous once again.

1090–1310 These lines are part of the line editing subroutine. They are used when changing a word or phrase. When changing a word or phrase, it is important to include the comma, period, or other punctuation as part of the word or phrase. Otherwise the program will not recognize it. There must be spaces on either side of the word or phrase that you wish to replace, unless it is at the beginning or end of the sentence. This is done to prevent the computer from changing parts of words that are themselves words, like AT in the word SAT. You would like to change AT to ON, not SAT to SON, for example.

Program Operation. After entering the program, type RUN. You will notice that the characters will shift to the upper/lowercase character set; the screen will blank and then display a menu screen in white letters. The Commodore will be locked in the upper/lowercase character set mode. Whatever you type will be in lowercase unless you press the SHIFT key while typing the letters.

You may simply start typing away and an underline character will act as the cursor. You can delete characters by pressing the DEL key, as long as the cursor has not jumped to the next set of lines. Then you will have to use the editing routine to change things. Don't be concerned if the words are split up going from the first 40 characters of the text to the last forty. They will be printed out correctly on the printer. When you press the space bar after you get past the 65th character in a set of two lines, the cursor will jump to the next line. You can also make it jump to the next line by pressing the RETURN key. If you keep pressing the RETURN key, you effectively create blank lines. After you get to the bottom of the screen, you will notice that the screen will automatically scroll upward.

Start new paragraphs by pressing the RETURN key and then spacing in about five spaces. Then continue typing. You can change the mode by pressing the RETURN key and then one of the function keys (f1 — f8). f2 is usually a safe key to press because it merely displays the text. After you press one of the f keys, the program will cycle to the main menu again. You can press another f key or continue entering the text by simply typing.

There is one problem that the Commodore has. As the memory gets full, the computer will stop from time to time, just for a few moments, but it can be a little aggravating. Fortunately the keyboard buffer will hold about 10 characters, so usually what you type in will not be lost, unless you are a fast typist or the memory is quite full. This is the only real problem that I've noticed with the system. The lack of 80 columns on the screen is a problem that can be lived with unless you have some software or hardware that can correct the problem. Otherwise the program is quite handy to use.

If you have a disk drive, you might consider having the computer save text on disk. You might even change the program so that all the text is saved directly on disk as it is typed. This might solve the problem of the keyboard delay. You might change the program to include graphic characters. You might also consider a 40-column screen editor where you can change any word or phrase on any line so long as it is still displayed on the screen. There are many possible improvements that can be made to this program.

Chapter 7

Other Areas in Artificial Intelligence

This final chapter will tie up some loose ends and present some new topics including LISP, an artificial intelligence computer language, robotics, creativity in computers, an artificial intelligence operating system, the limits of artificial intelligence, and computers specifically designed for artificial intelligence purposes.

LISP: AN ARTIFICIAL INTELLIGENCE LANGUAGE

LISP is a LISt Processing language used widely on some of the large computers involved in artificial intelligence. In fact a version of ELIZA is written in LISP. Most microcomputers have been unable to run the LISP language because it required such a large amount of memory. It is now available for some larger personal computers. It remains to be seen whether it will be available for the Commodore line of microcomputers.

LISP was developed by John McCarthy from an original language called IPL. LISP's usefulness soon became apparent, and now there are several dialects of the language around. The language is the preferred language of artificial intelligence programmers. Its speed of processing is much faster than that of BASIC. The structure of the language is unique. The language consists of atoms and lists. There is no distinction between programs and data. LISP is very useful in recursion problems and should be considered by anyone serious about artificial intelligence.

ROBOTICS

Since the introduction of the first small scale microprocessor controlled robot, there has been much interest in robotics. Some have speculated that robotics might become an industry as important as the microcomputer industry. This remains to be seen. Nevertheless, a discussion of robotics would be appropriate in a book on artificial intelligence.

Introductory Robots

Let's talk about some of the robots that can be used at school or home for learning about robotics. Let's look at some that are available now and some that may be available later.

I think that personal robotics really came into being when the HERO 1, Model ET18 was introduced by the Heath Company. This robot has a 6808 CPU with 4K RAM and 8K ROM. It can detect sound and light. It can detect objects with a range resolution of 1/4 inch to 8 feet. It can detect motion. It has speed synthesis and a calendar clock. It has wheels and an arm to manipulate objects. These features made this robot a real pioneer in the area of personal robotics.

Other robots on the market include "turtles," arms, and boxes. The RHINO, by Sandhu Machine Design, and the MINIMOVER, by Microbot Inc., are similar in that they consist of an arm that can grasp and manipulate small objects. Microbot also makes a TEACHMOVER robot arm machine. Terrapin Inc. makes two turtles, the TASMAN TURTLE and the TURTLE II. These robots can draw turtle graphics on paper. Other robots include ITSABOX, by Technical Micro Systems Inc., and RB5X, by RB Robot Corp. These robots vary in price and features.

If the personal robot industry takes off like the microcomputer industry did, we might expect to see robots that would have larger memories and more capabilities. This may or may not happen. Small robots would probably be useful in small manufacturing plants, but their real usefulness in the home is in doubt. They might be useful as a mobile computer. I think the technology has to jump a bit more before they would be cost-effective for home use. They might be useful as part of a security system in the event of a burglary or fire. Maybe someone will invent one that will pick up the house or do windows.

Robotic Activities

There are several activities that robots can perform. Here are a few of them.

Motion in Several Directions. Robots like the HERO or the turtles can move around the room or draw figure on paper because their wheels are controlled by motors. The turtles are accurate because of their plotting application. Floor models are less accurate. They are intended to find their way around objects and to measure their distance from them.

Robots with arms can raise the arm with one or two elbows, grasp with two fingers, and rotate the hand at the wrist. Some can extend the arm and rotate it. All of the motions that we have mentioned can be programmed. The program might consist of a machine language program with angles and distances entered. Some robots can learn to perform certain activities. Some linear assembly jobs can be accomplished using robots with this kind of motion.

Speech Synthesis. Speech synthesis in robots is just an extension of speech synthesis on microcomputers. The same electronics is involved. Undoubtedly better and better speech synthesis modules that will be able to create natural sounding speech will be made in the future. Ideally a robot would be able to talk like a person and have a large vocabulary that it could use in an intelligent manner.

Speech Recognition. Some robots can be trained to perform certain tasks whenever they hear certain sounds. A practical robot would be one that could distinguish and interpret speech and perform new commands based on auditory commands. There are expensive robots that can interpret speech and respond to what is said to them. This is an area that will be interesting to watch in the future.

CREATIVITY IN COMPUTERS

Can computers be creative? Within certain limits a computer might be considered creative. If a computer can select certain things randomly and then determine if that combination of things made sense, then it might be able to be creative. Common sense is a rather elusive concept, however, and computers are best at operating within a limited discipline, it seems.

In the Perfect Logic program, the computer was able to draw certain conclusions from the information that was entered. In this sense the computer might be considered to be creative, especially if you didn't know the mechanics of the program. A computer might be programmed to search for the relationships between all the components stored in memory. This might add to the creativity of the computer. If a computer could be programmed to develop analogies, it might test those analogies. If they made sense, then the computer might also be considered creative.

AN ARTIFICIAL INTELLIGENCE OPERATING SYSTEM

What would happen if you could get your hands on a computer that was constructed especially with artificial intelligence in mind? It would be interesting to have a computer that would operate in machine language and could interpret, store, and correlate information from English or any other language, as well as deal with math or other disciplines. Such a computer would be able to respond like a person might. This proposed computer would certainly be possible someday. The hardware is available today. The software end of it would require some careful planning before it could be feasible. Such a computer might be more precisely trained than programmed. I think that a computer with an artificial intelligence operating system is possible and even likely in the near future.

Parallel processing would contribute to the production of such a machine. Parallel processing means that several operations are performed at the same time. Most computers today have only one CPU that controls a serial schedule of operations. Even multitasking computers really simply divide up the time of the CPU and then perform a certain number of operations for each task. Imagine a computer built around arrays of CPU, each one controlling certain operations within its domain. Think of an electronic spreadsheet where an individual CPU controls the value of each cell and you might understand how such a machine might operate. I have no doubts that we will see parallel processing on microcomputers in the not too distant future.

LIMITS TO ARTIFICIAL INTELLIGENCE

I really think that there are no limits to artificial intelligence in computers. The human programming aspect seems to be the limiting factor. Memory size is also a limiting factor. I think that when optical disk storage, especially erasable optical disk storage, comes of age and is affordable, and when parallel processing becomes common, artificial intelligence will really flourish. There may come a time when computers will be able to program themselves. (After all, when you grew to a certain age, you could read and understand books by yourself and didn't need someone to explain everything to you.)

I really don't think that there are any real limits to artificial intelligence. It is important to remember that artificial intelligence is artificial. People and computers don't work the same way, and we should not forget that computers, no matter how remarkable, are still simply electronic machines.

ARTIFICIAL INTELLIGENCE COMPUTERS

At that time of this writing there are no true artificial intelligence microcomputers on the market. Texas Instruments proposed THE ANSWER, which could be programmed to interpret English. A computer like Apple's Macintosh goes a ways in the direction of artificial intelligence by making computers easier to program and operate. Hardware and software that allows for speech recognition are also a push in that direction. I think that someday there will be a truly artificial intelligent computer, one that will remember anything, that can communicate in natural language and can be programmed by verbal commands. Someday science fiction will be reality.

A BONUS GRAPHICS PROGRAM

The final bonus program shown in Listing 7-1 allows you to convert your computer into a drawing board. You use a joystick as a pencil or eraser. Actually you are bit-mapping the screen and are using a sprite in the shape of an arrowhead to draw or erase the pixels. By pressing the f1 key you can switch between the drawing and the erasing modes. By pressing the button on the joystick, you actually create or erase the pixels at the end of the arrowhead. By moving the arrowhead to the top left corner of the screen and pressing the joystick button, you can erase the entire screen. By moving the arrowhead to the rectangular figure to the right of the leftmost top figure, you can transfer the image on the screen to paper by using an Epson MX series or compatible printer. (I used a Cardco +G Printer Interface and an Epson MX-70 Printer.) The screen dump is very fast—about two minutes for whatever is on the screen. Other printers may work, but some printers use only 7 bits, so you may have to change the screen dump subroutine in the program (lines 600-700). I used a very inexpensive joystick and plugged it into CONTROL PORT 2 on the Commodore 64. It responded quite well. Figure 7-1 is a simple example of some of the things that you can draw with the joystick. The figures print sideways on the printer to give realistic proportions.

Listing 7-1 Graphics Maker

```
0 :::::::::::::::::::::::::::::::::::::::
1 REM     *** GRAPHICS MAKER ***
2 REM WRITTEN BY TIMOTHY J. O'MALLEY
3 REM COPYRIGHT 1984, TAB BOOKS INC.
4 REM (WRITTEN FOR THE COMMODORE 64)
5 :::::::::::::::::::::::::::::::::::::::
9 REM       *** INITIALIZE ***
10 FOR I=32768 TO 32795:READ N:POKE I,N:NEXT
15 FOR I=32799 TO 32823:READ N:POKE I,N:NEXT
20 BASE=16384:GOSUB 510:PRINT "(CLR)";
```

```
30 POKE 56578,PEEK(56578)OR3:REM SET OUTPUT BITS
40 POKE 56576,(PEEK(56576)AND252)OR2:REM BANK 1 STARTS AT 16384
50 S=53265:POKE S,PEEK(S)OR32:REM SET BIT MAP MODE
60 POKE S+7,(PEEK(S+7)AND15)OR128:REM RELOCATE SCREEN AT 24576-25599
70 POKE 32769,100:POKE 32775,96:POKE32779,14:SYS 32768:REM CLEAR
   SCREEN
80 POKE 32769,96:POKE 32775,64:POKE 32779,0:REM SET FOR CLEARING BIT
   MAP
90 DIM ZZ(255):REM MAKE ARRAY FOR SCREEN DUMP
95 FORI=0TO255:READ ZZ(I):NEXT
99 :
100 REM      *** MAKE SPRITES ***
110 FOR I=25600 TO 25662:READ N:POKE I,N:NEXT:X=200:Y=100
120 V=53248:POKE 25592,144:POKE V+39,1:POKE V,X:POKE V+1,Y
130 FOR I=25664 TO 25726:READ N:POKE I,N:NEXT
140 POKE 25593,145:POKE V+40,1:POKE V+2,25:POKE V+3,50:POKE V+21,3
199 :
200 REM     *** OPERATE JOYSTICK ***
205 P=0
210 GETP$:IFP$=CHR$(133) THEN P=NOTP
211 IFP$=CHR$(134)THEN GOSUB 800:REM SAVE BIT MAPPED SCREEN
212 IF P$=CHR$(135) THEN GOSUB 900:REM RETRIEVE DISK FILE OF BIT MAP
215 JV=PEEK(56320):FR=JV AND 16:JV=15-(JV AND 15)
220 ON JV GOSUB 300,310,320,330,340,350,360,370,380,390
230 IF FR<>16 THEN GOSUB 395
240 HX=INT((X+25)/256):LX=X+25-256*HX:POKE V,LX:POKE V+16,HX
250 POKE V+1,Y+50:GOTO 210
290 :
299 REM     *** JOYSTICK DIRECTIONS ***
300 Y=Y+(Y>0):RETURN:REM UP
310 Y=Y-(Y<199):RETURN:REM DOWN
320 RETURN
330 X=X+(X>0):RETURN:REM LEFT
340 Y=Y+(Y>0):X=X+(X>0):RETURN:REM UP & LEFT
350 Y=Y-(Y<199):X=X+(X>0):RETURN:REM DOWN & LEFT
360 RETURN
370 X=X-(X<319):RETURN:REM RIGHT
380 Y=Y+(Y>0):X=X-(X<319):RETURN:REM UP & RIGHT
390 Y=Y-(Y<199):X=X-(X<319):RETURN:REM DOWN & RIGHT
391 IF X>23 THEN 410
392 IF Y>12 THEN 410
395 IF X<10 AND Y<13 THEN 510
396 IF X<24 AND Y<13 THEN 610
399 :
400 REM      *** PLOT POINTS ***
410 BY=BASE+320*INT(Y/8)+8*INT(X/8)+(Y AND 7)
415 IF P THEN 460
420 POKE BY,PEEK(BY)OR(2↑(7-(X AND 7))):RETURN
449 :
450 REM      *** UNPLOT POINTS ***
460 POKE BY,PEEK(BY)AND(255-2↑(7-(X AND 7))):RETURN
499 :
500 REM *** CLEAR BIT MAPPED SCREEN ***
510 SYS 32768:RETURN
```

```
599 :
600 REM   *** HIGH-RESOLUTION SCREEN DUMP ***
610 OPEN 4,4,4:CMD4
620 PRINTCHR$(27)CHR$(65)CHR$(8);
630 FOR L=BASE+312 TO BASE STEP -8
640 PRINTCHR$(27)CHR$(75)CHR$(200)CHR$(0);
650 FOR B=0 TO 7680 STEP 320:N=L+B
670 FOR P=0 TO 7:PRINTCHR$(ZZ(PEEK(P+N)));
690 NEXT:NEXT:PRINT:NEXT
700 PRINT#4:CLOSE4:RETURN
999 REM    *** RETURN TO NORMAL ***
1000 POKE S+7,PEEK(S+7)AND127:REM RELOCATE NORMAL SCREEN
1010 PRINT "(CLR)";
1020 POKE S,PEEK(S)AND223
1030 POKE V+21,0:REM TURN OFF SPRITES
1100 REM DATA FOR MACHINE LANGUAGE ROUTINE
1110 DATA 162,96,169,,133,251,169,64,133,252,169,,160,,145
1120 DATA 251,200,192,,208,249,230,252,228,252,208,239,96
1130 DATA 173,,220,170,41,16,168,138,41,15,141,27,160
1140 DATA 169,15,237,171,160,141,28,160,140,29,160,96
1178 :
1179 REM DATA FOR ZZ ARRAY FOR SCREEN DUMP
1180 DATA 0,128,64,192,32,160,96,224,16,144,80,208,48,176,112,240,8,
     136,72
1181 DATA 200,40,168,104,232,24,152,88,216,56,184,120,248,4,132,68,196,
     36
1182 DATA 164,100,228,20,148,84,212,52,180,116,244,12,140,76,204,44,
     172,108
1183 DATA 236,28,156,92,220,60,188,124,252,2,130,66,194,34,162,98,226,
     18,146
1184 DATA 82,210,50,178,114,242,10,138,74,202,42,170,106,234,26,154,90,
     218
1185 DATA 58,186,122,250,6,134,70,198,38,166,102,230,22,150,86,214,54,
     182,118
1186 DATA 246,14,142,78,206,46,174,110,238,30,158,94,222,62,190,126,
     254,1,129
1187 DATA 65,193,33,161,97,225,17,145,81,209,49,177,113,241,9,137,73,
     201,41
1188 DATA 169,105,233,25,153,89,217,57,185,121,249,5,133,69,197,37,
     165,101
1189 DATA 229,21,149,85,213,53,181,117,245,13,141,77,205,45,173,109,
     237,29,157
1190 DATA 93,221,61,189,125,253,3,131,67,195,35,163,99,227,19,147,83,
     211,51
1191 DATA 179,115,243,11,139,75,203,43,171,107,235,27,155,91,219,59,

1192 DATA 251,7,135,71,199,39,167,103,231,23,151,87,215,55,183,119,247,
     15,143
1193 DATA 79,207,47,175,111,239,31,159,95,223,63,191,128,255
1199 :
1200 REM    *** DATA FOR SPRITE 0 ***
1210 DATA 128,,,96,,,56,,,30,,,15,128,
1220 DATA 7,224,,3,248,,1,192,,,192,,,64,
1230 DATA ,,,,,,,,,,,,,,,,,,,,
1240 DATA ,,,,,,,,,,,
1299 :
```

```
1300 REM    *** DATA FOR SPRITE 1 ***
1310 DATA 255,231,255,128,36,1,128,36,125
1320 DATA 128,36,1,128,37,253,128,36,1,128,37,253
1330 DATA 128,36,1,128,37,253,128,36,1,128,37,241
1340 DATA 128,36,1,128,36,1,255,231,255
1350 DATA ,,,,,,,,,,,,,,,,,,,,,
```

Let's look briefly at the program. Lines 0-5 identify the program. Lines 9-99 initialize the program by reading data, changing screen modes, and dimensions an array. Lines 100-199 read the data and define the sprites used in the program. Lines 200-290 operate the joystick. You will notice that the f3 key can be used to save the bit map if you define a subroutine in lines 800-899. Likewise if you define a data retrieval subroutine in lines 900-998, you can use f5 to retrieve that data. Lines 299-399 calculate the position of the pixel to be plotted or erased. Lines 400-449 plot the points by turning on the pixels of the bit-mapped screen. Likewise lines 450-499 unplot the points by resetting pixels of the screen. Lines 500-599 clear the entire screen by calling a machine language subroutine that the program located at 32768. Lines 600-700 form the high-resolution screen dump that will work with some printers. You might want to substitute a screen dump subroutine that will work with your particular printer. Lines

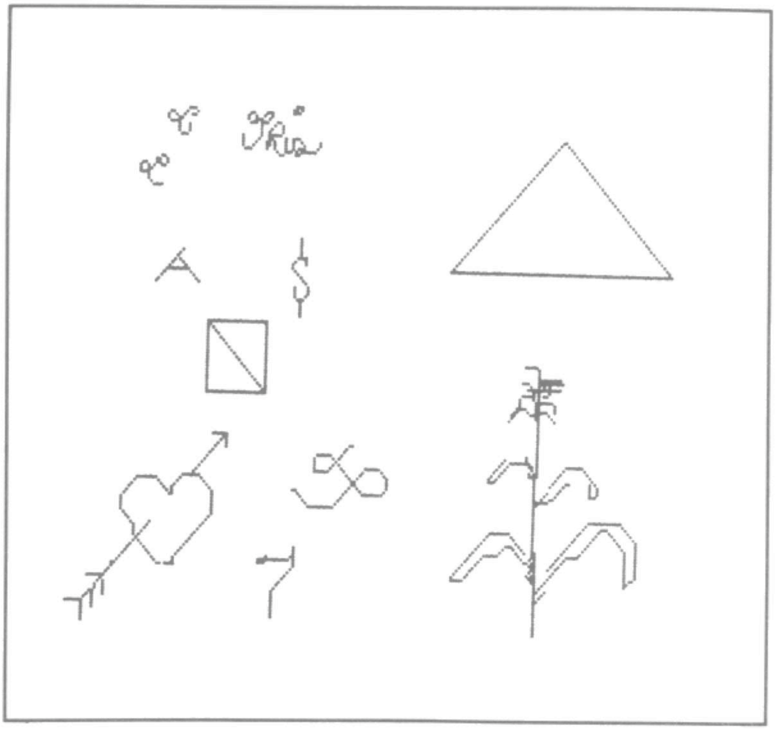


Fig. 7-1. A screen dump of graphic images.

999-1030 are lines that restore the computer to normalcy. You will have to STOP the program by pressing the RUN/STOP key and typing GOTO 1000. It may be simpler to turn the power off and then on to restore everything when you are done. The rest of the lines are data for the machine language subroutine, the screen dump, and the sprites.

Glossary

Address: A specific location in memory, from 0-65535 in decimal or 0000-FFFF in hexadecimal for 8-bit CPU computers.

Algorithm: The specific set of rules or operations to solve a problem.

Alphabetic: Consisting of the letters A-Z and blanks.

Alphanumeric: Consisting of letters, numbers, and special characters, the keys of the keyboard.

And: A boolean logical operator that makes an expression true if all terms are true.

Array: A data structure made of numerics or literals.

Artificial Intelligence: Any method that attempts to simulate the cognitive processes generally associated only with human thought.

ASCII: American Standard Code for Information Interchange, a code representing the alphanumeric characters in computing, 0-255.

BASIC: Beginner's All-purpose Symbolic Instruction Code, a high-level computer language commonly used in microcomputers.

Baud: A rate of data transfer along a serial line, first used with teletype machines.

Behavior: An overt action expressed by an organism.

Binary: Consisting of only two possible states, 0 or 1. Zero usually represents a false or off condition and 1 represents a true or on condition. Computers use the binary system.

Bit: A binary digit, either 0 or 1.

Bootstrap: A method of allowing a computer program to change itself.

Branch: A statement, like GOTO, that changes the flow of a program.

Bug: A flaw in a computer program.

Byte: A location in memory that can store an integer from 0-255, an 8-bit number.

Chip: Slang for an integrated circuit used in RAM or a CPU.

Compiler: A program that changes a high-level program to a low-level one.

CPU: The Central Processing Unit, the arithmetic, timing, and logic unit of the computer. It is the real "workhorse" of the computer.

Data: Any computer information.

Disk: A round magnetic storage medium.

File: A collection of data stored on tape or disk.

Firmware: Software stored in ROM.

Hard copy: A printout on paper.

Hardware: The physical machine.

Heuristic: A rule-of-thumb or any method that can allow for the faster solution of any problem by eliminating nonsense.

Hexadecimal: The base 16 numbering system, which uses 0-9 and the letters A-F.

Input: What you enter into the computer program.

Integer: Any whole number.

Interpreter: A program that converts each statement in a program in a high-level language to a token or machine language code as the program is run.

Keyboard: Any collection of typewriter-like keys used to enter input to a computer.

Literals: Any group of characters within quotes or assigned to a string.

Loop: A sequence of computer instructions that is repeated.

Machine language: The actual code used by the CPU.

Mass storage device: A device, like disk or tape, used to store data.

Memory: Storage locations in a computer.

Microcomputer: A personal computer using a microprocessor.

Microprocessor: The CPU on a chip, like the 6502 or the 6510.

Numeric: Consisting only of digits 0-9, the minus sign, the plus sign and the decimal point.

OR: A boolean logical operator that makes an expression true if any term is true.

Parallel interface: An interface that send all the bits of the byte at once, as opposed to a serial interface.

Port: The connection point between the CPU of the computer and the peripheral.

Printer: A device that makes hard copies.

Program: The list of instructions that controls a computer.

RAM: Random Access Memory, the memory in a computer that can be actively changed.

ROM: Read Only Memory, the memory of the computer that can not be changed, usually containing the operating system of the computer.

Serial interface: An interface that sends the bits of a byte in sequence along one data line.

Software: Programs and data.

String: A variable, such as S$, that contains data as a collection of characters.

Subroutine: A small program contained within a larger program.

Turing machine: An automatic device that is the logical basis of every CPU, theorized by A. M. Turing.

Variable: A quantity specified by a name that has values that can change.

Index

Edited by Marilyn L. Johnson

Artificial Intelligence Projects for the Commodore 64™

by Timothy J. O'Malley

Discover a whole new dimension in your C-64's programming abilities!

If you're tired of ordinary computer games . . . if you're looking for something exciting and different to do with your C-64 . . . here's the answer! It's a whole collection of artificial intelligence (AI) projects designed to tap your micro's real problem-solving capabilities for both practical and entertainment applications.

Leading off with a definition of artificial intelligence and an overview of AI concepts, the author provides 16 ready-to-run programs in BASIC to illustrate your micro's cognitive powers. You'll cover tree searches (testing all possible solutions to a problem), hueristics (a modified trial-and-error technique), algorithms, and pattern searching/recognition routines.

You'll find out how to solve simple—and not-so-simple—puzzles like Towers of Hanoi and the Knight's Tour of the Chessboard . . . explore concepts of animal behavior and how it can be simulated . . . analyze how natural language can be recognized and acted on by the computer . . . simulate an actual human-machine conversation . . . and use an interactive routine that allows your micro to make deductions through clever application of set theory. There's even a program that allows your micro to write its own program modifications!

And, as an extra bonus, the author has included a functioning word processing program (which he used to write this book's manuscript) and a graphics program that lets you draw on the screen with a joystick.

Totally fascinating and packed with techniques that will help you improve all your BASIC programming practice, this is a sourcebook that will open a whole new dimension in your computer usage!

Timothy J. O'Malley is a writer and programmer whose experience spans both mainframe and microcomputer experiments in artificial intelligence.

TAB TAB BOOKS Inc.
Blue Ridge Summit, Pa. 17214

FPT > $12.95

ISBN 0-8306-1883-X

PRICES HIGHER IN CANADA

1245-0285